华中科技大学

课 程 实 验 报 告

课程名称：C++程序设计

实验名称：面向对象的公交地图导航

院　　系：计算机科学与技术

专业班级：CS2006

学　　号：U20201547

姓　　名：杨释钧

指导教师：纪俊文

2021 年 12 月 18 日

目录

1 需求分析

1.1 题目要求

假定所有公交车辆从起点到终点都是双向非环路的，且双向线路的所有停靠站点都对应相同。设有 M 路公交路线，第 j 路公交路线有 N_j 个站点。所有公交线路累计共有 S 个站点，第 k 个站点的坐标为（X_k，Y_k），所有坐标均以米为单位标注。邻近站点之间的距离指的是站点坐标之间的欧几里得距离。现有一人处于起点坐标（X_b，Y_b），此人需要步行到最近站点乘车，下车后要步行到达的终点坐标为（X_e，Y_e），而他特别不愿意走路，能坐公交就尽量坐公交。假定公交转乘时的步行距离为 0，试编程求他从起点（X_b，Y_b）到终点（X_e，Y_e）的乘坐线路。建立模型时需要考虑如下几种情形：（1）最少转乘；（2）最短距离。

所有公交线路的站点坐标存放于“stops.txt”文件，其中第 1 行为站点总个数，第 2 行为第 1 个站点的坐标，第 3 行为第 2 个站点的坐标，以此类推。可用图形化的界面显示站点及公交线路。“stops.txt”文件的内容如下。

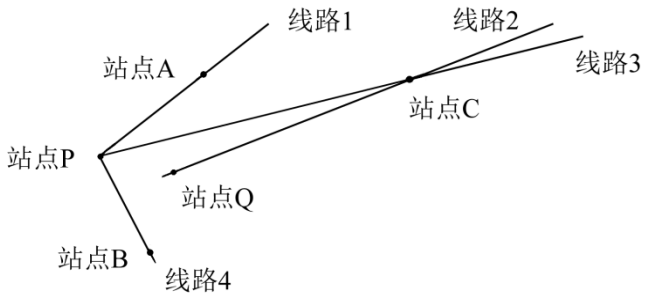
```
39
235     27
358     29
480     34
155     36
222     64
282     62
413     60
457     63
483     60
560     69
131     87
349     61
314     97
420    107
487    125
620    107
666     79
186    107
270    120
350    141
383    148
370    164
```

442 179
496 171
555 167
651 155
775 184
678 272
208 156
296 161
356 190
493 202
490 229
504 262
457 269
249 196
155 190
103 171
112 241

所有公交线路信息存放于“lines.txt”文件，其中第 1 行为公交线路总数，第 2 行为每条公交线路的站点总数，第 3 行为线路 1 经过的站点编号（对应站点坐标参见“stops.txt”），第 4 行为线路 2 经过的站点编号，以此类推。“lines.txt”文件的内容如下。

6
13 11 8 9 7 7
1 6 13 20 22 21 14 8 3 9 15 24 32
4 5 6 12 7 8 9 10 16 26 28
11 18 19 20 21 23 33 34
38 37 36 31 23 24 25 26 27
2 12 13 19 29 37 39
30 31 35 33 25 16 17

采用 Dijkstra 最短路径算法是不合适的，因为它仅考虑了距离而未考虑公交线路行驶约束。如下图所示，对于某站点 P 周围的站点 Q，其欧几里得距离虽近，但可能需要很远的转乘才能到达。例如，从站点 P 出发，要从线路 3 经站点 C 转线路 2 才能到最近的站点 Q，因为站点 P 和站点 Q 之间无直达的公交线路。



通过类型抽象形成站点、公交线路、转乘站点、转乘线路、转乘矩阵、公交系统等类，输入上述文件初始化站点和线路对象，然后通过图形化的界面显示站点和线路地图。用户用鼠标左键在地图上设定起点、鼠标右键确定终点，按照设定的最少转乘或者最短距离选项规划出从起点步行到最近站点上车、到离终点最近站点下车步行到终点的线路，在地图上用不同颜色显示规划线路若干秒，然后消除并恢复原始地图线路的颜色。

假设某个机构或单位的信息存储在“organization.txt”文件，第 1 列存放单位名称，第 2 列单位坐标。“organization.txt”文件中的格式如下，可自己添加更多单位或坐标。

华中科技大学　990, 370

华乐山庄　500, 340

光谷中心花园　631, 367

光谷街北路　766, 472

用户可输入“华科大”或“华中科大”查询其所在位置，通过最大公共子串算法进行模糊匹配，找到“华中科技大学”及其坐标。在确定始发单位坐标和终点单位坐标后，按照设定的最少转乘或者最短距离选项规划线路，并在地图上用不同颜色显示规划线路若干秒，然后消除并恢复原始地图线路的颜色。

提示：可以构造公交线路转乘矩阵 A（即 A^1）。假定有 5 条公交线路，若线路 i 和线路 j（i≠j）有 r 个转乘站点（即 r 种走法），则矩阵 A^1 的元素 $a^1[i,j]=r$；如 i=j 则规定无须转乘，即有 $a^1[i,i]=0$。则对于上述前 5 条公交线路，从公交线路 i 一次转乘到线路 j，可得如下转乘矩阵 A^1。

0　3　3　1　0

3　0　3　2　1

3　3　0　0　1

1　2　0　0　1

0　1　1　1　0

由上述转乘矩阵 A^1 可知，无法从线路 1 转乘到线路 5，因为 $a^1[1,5]=0$。可以尝试从线路 1 转乘其他线路，再从其他线路转乘线路 5，即从线路 1 经过两次转乘到线路 5。这只需要进行矩阵乘法运算 $A*A=A^2$，从线路 1 经过两次转乘到线路 5，可从 A^2 中得到共有 $a^2[1,5]$种走法。

$$a^2[1,5]=\sum_{k=1}^{5} a^1[1,k]*a^1[k,5]$$

由上述转乘公式，可计算得到 A^2，最后设置 $a^2[i,i]=0$，修正 A^2 使同一线路不用转乘。修正后的 A^2 如下。

0　11　9　6　7

11 0 10 4 5

9 10 0 10 3

6 4 10 0 2

7 5 3 2 0

由上述矩阵可知 a^2 [1,5]=7，即从线路 1 经过两次转乘到线路 5 共有 7 种走法：（1）从线路 1 到线路 2 共有 3 个转乘点，再从线路 2 经唯一转乘点至线路 5，一共有 3 种转乘方法；（2）从线路 1 到线路 3 共有 3 个转乘点，再从线路 3 经唯一转乘点至线路 5，一共有 3 种转乘方法；（3）从线路 1 经唯一转乘点至线路 4，再从线路 4 经唯一转乘点至线路 5，一共有 1 种转乘方法。

依此类推，可以得到 A*A*A（即 A^3，反映从线路 i 经过 3 次转乘到线路 j 共有几种转乘走法）。对于本题来说，由于总共只有 M=7 条公交线路，故无重复公交的转乘最多只能转乘 M-1 次，所以只需计算到 $A^{M-1}=A^6$ 为止。任意两条线路之间 1 次、2 次……6 次转乘的走法共有 A^+种，$A^+= A^1+ A^2+ A^3+ A^4+ A^5+ A^6$。

上述 A^+运算是一种修正的矩阵“闭包”运算，可以抽象成矩阵类的一个运算。上述 A^+运算只计算了有几种转乘走法，当然，还可以同步 A^n 建立另外一个矩阵，对应元素为某种链表指针类型，用于记载对应转乘线路和转乘站点。

得到上述转乘“闭包”矩阵 A^+以后，只需要搜索离起点最近的站点（可能不止一个站点），找到最近站点所在的线路 i（可能不止一条），并搜索离终点最近的站点（可能不止一个站点），找到最近站点所在的线路 j（可能不止一条），然后在 A^+中找出 a[i, j]所描述的所有转乘线路和转乘站点即可。利用站点坐标可以得到两两站点之间的距离，并利用 A^+分别得到如下两种情形的最优转乘方案：（1）最少转乘；（2）最短距离。

1.2 需求分析

本函数用于实现公交路线导航。程序能实现根据提供的路线和站点信息，由用户在地图选择起点和终点，用户可选择根据最少转乘或最短距离导航，系统会给出相应的路线实现导航，同时，用户也应可以自己输入地点名称进行查询，即需要实现模糊匹配的功能并进行查找的功能。

2 系统设计

2.1 概要设计

本程序主要有三个板块，即作为逻辑层的用于计算转乘闭包矩阵的 logiclayer 板块和作为图形层用于 QT 界面的 MAP 板块以及 main 函数板块。

logiclayer 板块可分为 8 个小板块：描述公交站点的 STOP 板块，描述公交路线的 LINE 板块，描述路线转乘的 TRAN 板块，描述转乘路径的 ROUTE 板块，描述记录转乘次数和路线的 NODE 板块，描述矩阵闭包运算的 TMAP 板块，记录地点名称及坐标的 organization 模块和描述地理信息系统的 GIS 类。

MAP 板块可分为 4 个小板块：用于公交站点及公交路线文件的输入或加载、获取功能公交站点坐标信息以及公交路线的所有站点信息的 QtWidgetsFL 板块，用于鼠标指针经过公交站点时提供提示信息的 QtTipsDlgView 板块，用于作为模糊匹配界面的 MainWindow 模块，用于负责公交站点及公交路线的加载以及最少转乘和最短距离转乘方案的查询的 twnlt 板块。

main 函数板块用于启动用户主窗口。

模块关系如图 2.1 所示。

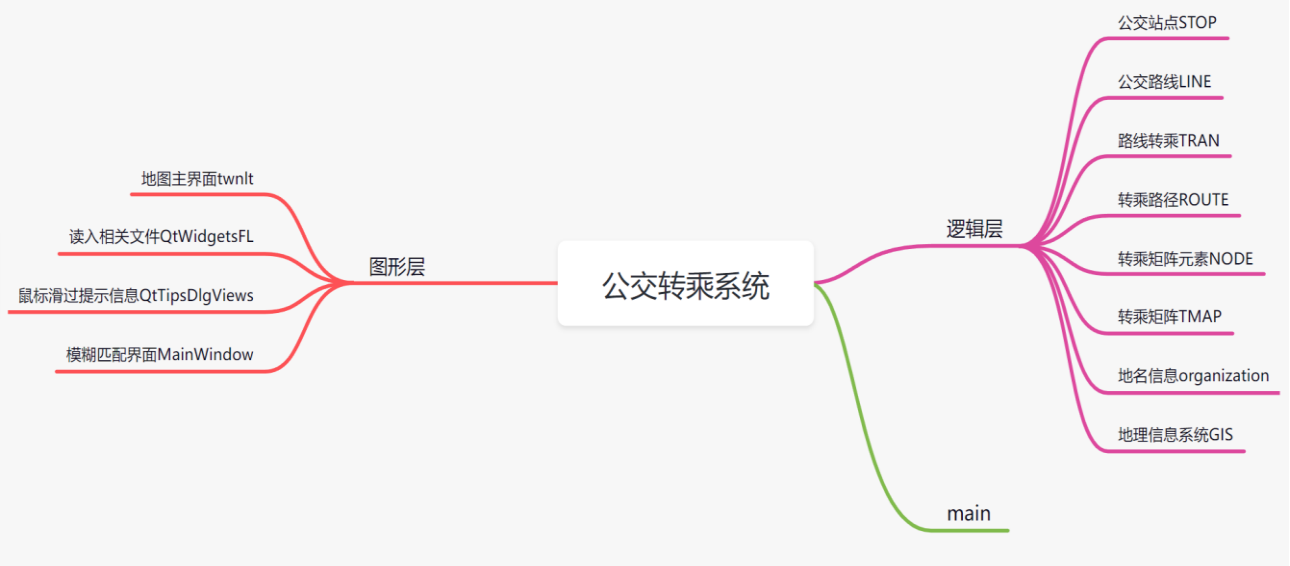


图 2. 1

实验的总体流程为启动用户可视化界面，读入站点和路线以及相关的地名信息，在地图上选择起点和终点，然后选择路线生成方式，由此得到导航路线，同时用户也可以选择自己输入地名，简易流程图如图 2.2 所示。

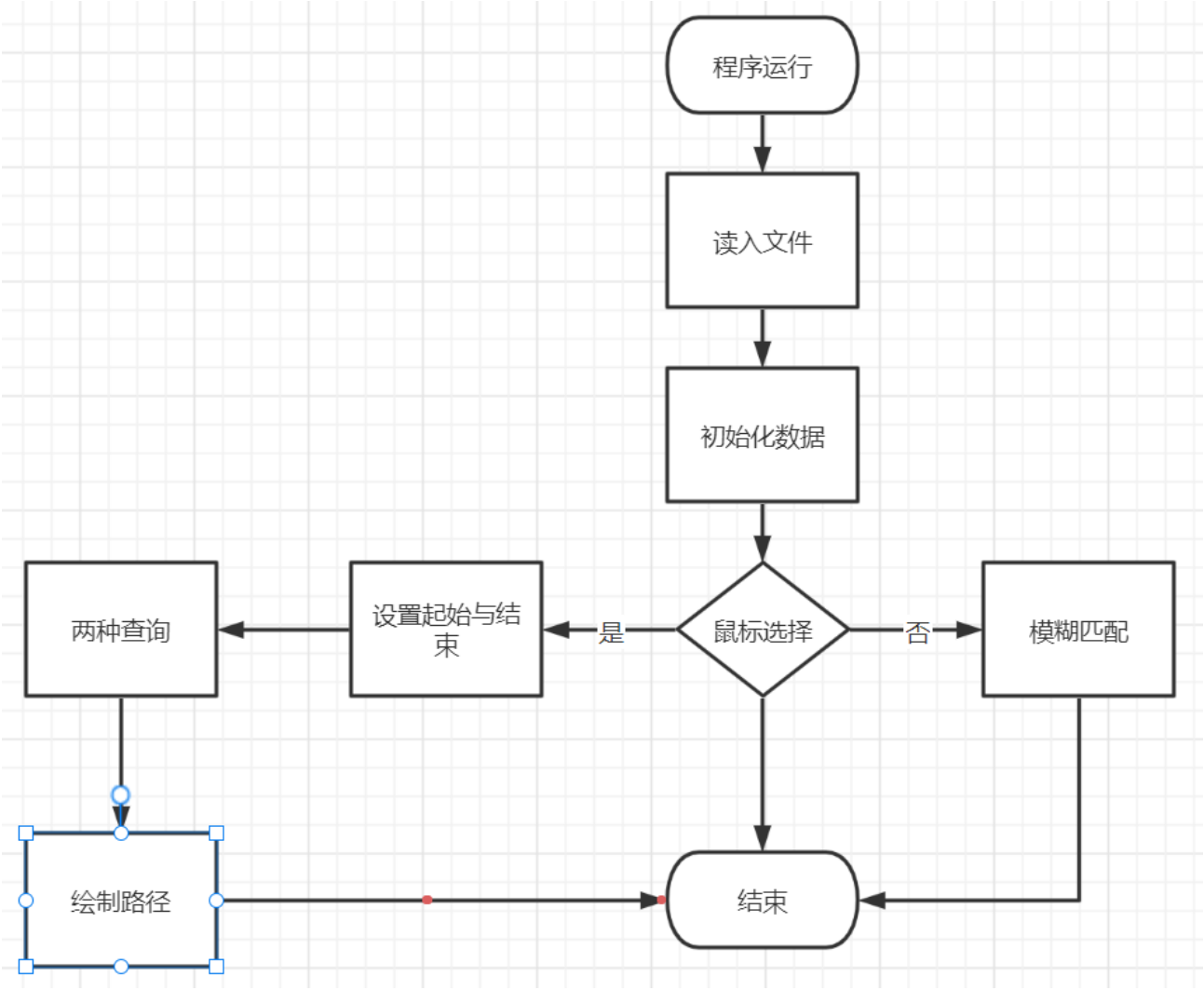


图 2.2

2.2 详细设计

1. logiclayer 板块。

本版块的主要功能是实现公交转乘矩阵的闭包运算。主要是分别为每种信息建立对应的类，然后在这些类的基础上完成对应的操作。

STOP 板块

本版块用于建立描述公交站点信息的 STOP 类。

STOP 类的结构为：

```
class STOP {                                    //描述一个公交站点
    int numb;                                   //所有公交站点编号
    int x, y;                                   //公交站点坐标
public:
    STOP(int n = 0, int x = 0, int y = 0);
```

```
    virtual int& X();
    virtual int& Y();
    virtual int& N();
    };
```

包含公交站点编号 numb 和坐标 x，y。

功能：获取站点的坐标和编号。

算法：本类作为存储模块，无特殊算法。

LINE 板块

本板块用于建立描述公交路线的 LINE 板块。

LINE 类的结构为：

```
class LINE {                                          //描述一条公交线路
    const int numb;                                   //公交线路编号从1开始
    int* const stop;                                  //公交线路上所有站点编号
    const int nofs;                                   //公交线路上站点数量
public:
    LINE(int numb = 0, int nofs = 0, int* stop = nullptr);
    LINE(const LINE& r);
    LINE(LINE&& r)noexcept;
    LINE& operator=(const LINE& r);
    LINE& operator=(LINE&& r)noexcept;
    virtual int has(int s)const;                      //若包含站点编号s，则返回线路中的
站次序号；-1表示未包含
    virtual int cross(const LINE& b)const;            //若两条公交线路相交，则返回1
    virtual operator int()const;                      //取公交线路编号
    virtual int NOFS()const;                          //取公交线路的站点数量
    virtual double dist(int d, int a)const;           //线路从站次d到站次a的距离
    virtual int& operator[](int x);                   //取线路某个站次的站点编号
    virtual ~LINE()noexcept;
    };
```

包含公交路线编号 numb，路线上所有站点编号 stop，公交路线上站点数量 nofs。

功能：判断是否包含某个站点编号，判断两条公交路线是否相交，取公交路线编号，求站点之间的距离，取路线某个站点编号。

算法：本类作为存储模块，无特殊算法。

TRAN 板块

本板块用于建立一次转乘的 TRAN 类。

TRAN 类结构为：

```
class TRAN {                                          //从线路from经站点编号stop转至线路to
    int from;                                         //现乘的公交线路号
    int to;                                           //需要转乘的公交线路号
    int stop;                                         //由stops. txt定义的站点编号
public:
    TRAN(int from = 0, int to = 0, int stop = 0);
    int operator==(const TRAN& t)const;
    virtual int& F();                                 //现乘的公交线路号
    virtual int& T();                                 //需要转乘的公交线路号
    virtual int& S();                                 //转乘点的站点编号
    };
```

功能：储存一次转乘的数据，包括转乘站点的索引，转乘前的线路和转乘后的线路。

算法：int operator==(const TRAN& t)const;

判断 from, to 和 stop 是否均相等，均相等返回 true，否则返回 false。该功能是为接下来的算法做准备的。

ROUTE 板块

本版块用于建立存储转乘路径的 ROUTE 类。

ROUTE 类的结构为：

```
class ROUTE {                                         //一个转乘路径
    TRAN* const tran;                                 //转乘路径上的所有转乘站点
    const int noft;                                   //转乘路径上的转乘次数
public:
    ROUTE(TRAN* tran = nullptr, int noft = 0);
    ROUTE(const TRAN& t);
    ROUTE(const ROUTE& r);
    ROUTE(ROUTE&& r)noexcept;
    virtual operator int()const;                      //得到转乘次数
    virtual int operator==(const ROUTE& r)const;
    virtual ROUTE operator *()const;                  //约简：去掉重复的公交转乘线路
    virtual TRAN& operator[](int);//一条路径上的所有转乘站点
    virtual ROUTE operator+(const ROUTE& r)const;// 转乘路径连接
    virtual ROUTE& operator=(const ROUTE& r);
    virtual ROUTE& operator=(ROUTE&& r)noexcept;
    virtual ROUTE& operator+=(const ROUTE& r);
    virtual ~ROUTE()noexcept;
```

```
    virtual int print()const;
    //打印转乘路径
};
```

功能：储存一条路线上的所有转乘，并且实现两条转乘路线的连接和化简的运算。

算法：int operator==(const ROUTE& r)const;

当两条路线中每个换乘点的信息都完全相同时，这两条路线是相同的.

ROUTE operator *()const;

当这条路径中有两个 TRAN 完全相同的时候，说明从一个站点出发转了一圈又回到了这个站点，这就属于无效路径，要把这两个 TRAN 之间的换乘都删掉，这样就完成了路线的化简。

ROUTE operator+(const ROUTE& r)const;

首先判断两条路线是否可以相接（当上一条路线的最后一个 TRAN 的下一 to 等于这一条路线的第一个 TRAN 的当前 from 时，两条路线可以相接，否则不行）。

NODE 板块

本板块用于构造记录转乘次数和线路的矩阵元素的类 NODE。

NODE 类的结构为：

```
class NODE {//矩阵元素：记载的转乘次数和线路
    ROUTE* const p;//矩阵R*c个元素的转乘路径方案
    int n;//矩阵r*c个元素记载的转乘路径方案数
public:
    NODE(ROUTE* p, int n);
    NODE(int n = 0);
    NODE(const NODE& n);
    NODE(NODE&& n)noexcept;
    virtual NODE operator*()const;//矩阵元素约简：去掉转乘中的环
    virtual NODE operator+(const ROUTE& n)const;//元素增加路径
    virtual NODE operator+(const NODE& n)const;//元素路径增加
    virtual NODE operator*(const NODE& n)const;//元素路径转乘连接
    virtual NODE& operator=(const NODE& n);
    virtual NODE& operator+=(const NODE& n);//直接修改this指针指向的值
    virtual NODE& operator+=(const ROUTE& n);//直接修改this指针指向的值
    virtual NODE& operator*=(const NODE& n);//直接修改this指针指向的值
    virtual NODE& operator=(NODE&& n)noexcept;
    virtual ROUTE& operator[](int x);//获得第X个转乘路径
    virtual operator int& ();//返回可转乘路径数n
    virtual ~NODE()noexcept;
```

```
    virtual void print()const;//打印转乘矩阵元素
    };
```

功能：储存从某个站点到另一个站点的若干路径，通过+，*等实现站点到站点的路线计算与化简。

算法：operator+

将一个 NODE 或者 ROUTE 中的所有 ROUTE 全部加进原本的 NODE 中，再将相同的 ROUTE 进行约简，此操作类似集合的合并。

operator*

将一个 NODE 中所有 ROUTE 接到另一个 NODE 的所有 ROUTE 的后面，运算方式采用二重循环调用 ROUTE 的 operator+实现，此操作类似集合的笛卡尔积。

operator*()

将 NODE 中的 ROUTE 进行约简，调用 ROUTE 的 operator==函数判断两条 ROUTE 是不是同一条，如果是，删去其中一条即可。

TMAP 板块

本版块用于建立完成矩阵的闭包运算的类 TMAP。

TMAP 类的结构为：

```
class TMAP {//所有公交转乘元素的“闭包”矩阵
    NODE* const p;
    const int r, c;//行数和列数
public:
    TMAP(int r = 0, int c = 0);
    TMAP(const TMAP& a);
    TMAP(TMAP&& a)noexcept;
    virtual ~TMAP();
    virtual int notZero()const;//若出现不可到达站点，就返回0
    //以下函数返回最少转乘数的路径数，起点站次为b，终点站次为e, r存在10条路径
    virtual int miniTran(int b, int e, int& noft, ROUTE(&r)[100])const;
    //以下函数返回最短距离的路径数，起点站次为b, 终点站次为e, , r存在十条路径
    virtual int miniDist(int b, int e, double& dist, ROUTE(&r)[100])const;
    static double getDist(int b, int e, ROUTE& r);
    virtual NODE* operator[](int r);//得到矩阵某行r元素 的首地址
    virtual int& operator()(int r, int e);
    virtual TMAP operator*(const TMAP& a)const;//“闭包”运算：乘法
    virtual TMAP operator+(const TMAP& a)const;//"闭包“运算：加法
    virtual TMAP& operator=(const TMAP& a);
    virtual TMAP& operator= (TMAP&& a);
    virtual TMAP& operator+=(const TMAP& a);
```

```
    virtual TMAP& operator*=(const TMAP& a);
    virtual TMAP& operator()(int r, int c, const ROUTE& a);//将路径a加入矩阵元素中
    virtual void print()const;//打印转置矩阵
    };
```

功能：储存从一条公交线路到另一条公交线路的所有转乘方案，并根据矩阵的乘法和加法进行运算。

算法：minTran

首先找到途径开始站点 s 的线路 bls 和途径 t 的线路 els；对每个 bls 和 els 之间的转乘方案，可以方便的通过矩阵相应元素获取。将这些转乘方案的集合 rou 进行遍历（使用 NODE 重载的 operator[] 能容易地做到这一点），并比较两条路径之间的换乘数，总是选择最少换乘数的线路并保持更新结果 ROUTE 数组 r。

minDist

首先找到途径开始站点 s 的线路 bls 和途径 t 的线路 els；对每个 bls 和 els 之间的转乘方案，可以方便的通过矩阵相应元素获取。将这些转乘方案的集合 rou 进行遍历（使用 NODE 重载的 operator[] 能容易地做到这一点），并比较两条路径之间的路径总长，总是选择最短长度的线路并保持更新结果 ROUTE 数组 r。

organization 板块

本模块主要用来建立模糊匹配时需要用到的地点的存储类 organization。

organization 类结构如下：

```
class organization {
    const char *name;//本地点的名字
    int x, y;//本地点的坐标
public:
    organization(const char* name=nullptr, int x=0, int y=0);
    virtual int& X();//获取横坐标
    virtual int& Y();//获取纵坐标
    virtual const char* NAME();//获取名字
    void NAME(const char* &ch)//修改名字
    {
        name =new char[strlen(ch)+1];
        strcpy((char*&)name, ch);
        ch = nullptr;
    }
};
```

功能：存储模糊匹配时需要的地点信息。

算法：存储模块，无特殊算法。

GIS 板块

本板块用于构造描述地理信息的类 GIS。

GIS 类结构如下：

```
struct GIS {//描述地理信息系统的类
    static STOP* st;//所有公交站点数
    static organization* org;
    static LINE* ls;//所有公交线路
    static int ns, nl;//公交站数ns, 公交线路数
    static TMAP raw, tra;//原始转乘矩阵raw,“闭包”转乘矩阵
    static int obs;//GIS的对象数量
    static int numoforg;
    int start, end;
    bool flag = false;
public:
    GIS();
    GIS(const char* flstop, const char* flline, const char* florg=nullptr);//用站点及线路文件加载地图
    int miniTran(int fx, int fy, int tx, int ty, int& f, int& t, int& n, ROUTE(&r)[100]);
    int miniDist(int fx, int fy, int tx, int ty, int& f, int& t, double& d, ROUTE(&r)[100]);
    ~GIS();
};
```

功能：作为描述整个地理信息系统的类。

算法：作为存储模块，无特殊算法。

2. MAP 板块

本版块的主要功能是实现用户界面，完成该程序的可视层。使用户能够进行相关的可视化操作。

QtWidgetsFL 板块

本版块用于公交站点、公交路线及地点文件的输入或加载，获取功能公交站点坐标信息以及公交路线的所有站点信息，同时获取现有的地点的名称及对应坐标。

结构如下：

```
class QtWidgetsFL : public QWidget
{
    Q_OBJECT
public:
    QtWidgetsFL(QWidget* parent = Q_NULLPTR);
    ~QtWidgetsFL();
    QGraphicsView* parnt;
```

```
    void myShow(QGraphicsView* p);
private:
    Ui::QtWidgetsFL ui;
private slots:
    void inputStop();
    void inputLine();
    void checkFile();
    void inputOrg();
    };
```

功能简介：本版块通过 inputStop 函数读取 stops. txt 文件，通过 inputLine 函数读取 lines. txt 文件，通过 inputOrg 函数读取 Organization. txt 文件，获取公交站点坐标信息和公交路线的所有站点信息以及相关地点信息。然后在 checkFile 函数中将文本框中的文本转换成字符型指针，实现文件的打开，读入信息后设置显示的颜色，通过 parent->scene 函数在背景地图上画出公交站点和公交路线。

QtTipsDlgView 板块

本板块用于鼠标指针经过公交站点时提供提示信息。

```
class QtTipsDlgView : public QDialog
{
    Q_OBJECT

public:
    QtTipsDlgView(const QString& msg, QWidget* parent = Q_NULLPTR);
    ~QtTipsDlgView();
    void startTimer(int);
private:
    Ui::QtTipsDlgView ui;
    QTimer* m_pTimer;
    void initFrame(const QString& msg);
    };
```

功能简介：本版块用于当鼠标指针移动并经过公交站点时，可显示站点的提示信息，并在若干秒后自动消除提示。通过 setText 函数实现信息显示，而 m_pTimer->setSingleShot 则起到定时器的效果。

MainWindow 板块

本版块用来实现模糊匹配时用户输入地名的界面。

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
```

```
public:
    MainWindow(QStringList qstr, QWidget *parent = Q_NULLPTR);
    void FuzzySearch();
    void checkinput();
    ~MainWindow();

private:
    Ui::MainWindow ui;
    QStringList qstr;
    QString start;
    QString end;
    };
```

功能简介：模糊匹配界面由 QLineEdit 与 QCombox 组合实现，其中函数 FuzzySearch 用来实现简单地模糊匹配功能，checkinput 函数用来检查输入的地名是否正确，QStringList qstr 成员用来存储由地名文件输入的所有地名，QString start 用来获取起点的名称，QString end 用来获取终点的名称。

twnlt 板块

本版块用于负责公交站点及公交路线的加载，以及最少转乘和最短距离转乘方案的查询。

```
class twnlt : public QMainWindow
{
    Q_OBJECT
public:
    twnlt(QWidget* parent = Q_NULLPTR);
    ~twnlt();
private:
    Ui::twnltClass ui;
    QtWidgetsFL* fl;
    MainWindow* win;
    QTimer* m_Timer;
    QGraphicsItemGroup* gItem;
    void deleteItems();
protected:
    void closeEvent(QCloseEvent* event);
private slots:
    void loadmap();//加载地图
    void closewnd();//关闭界面
```

```
    void zszc();//查询最少转乘的路径方案
    void zdjl();//查询最短距离的路径方案
    void findinput();//进入模糊匹配界面，获取用户输入的地点名称
};
class MyScene :public QGraphicsScene
{
public:
    explicit MyScene(QObject* parent = 0);
    void stopLines(QGraphicsView*);
protected:
    QGraphicsView* qgv;
    void mouseMoveEvent(QGraphicsSceneMouseEvent* event);//捕获鼠标移动的动作
    void mousePressEvent(QGraphicsSceneMouseEvent* event);//捕获鼠标按压的动作
public slots:
};
class MyItem :public QGraphicsRectItem {
    int cx, cy;
    int cf;
    int cs;
    int bs[6];
public:
    MyItem(int x, int y, int f);
    MyItem& operator<<(int s);
    int operator()(int x, int y);
    int& x();
    int& y();
    int& f();
    int& c();
    int& operator[](int);
    int checkAllStops();
    void mousePressEvent(QGraphicsSceneMouseEvent* event);
};
class seItem {
    int cx, cy;
    int cf;
    int cs;
```

```
};
```

功能简介：本模块中包含了四个自定义的类，其中 twnlt 类是主界面类，包含了加载地图、最短路径、最少转乘等多个成员函数，定义捕获用户鼠标点击的地点的类 MyItem，其中的成员函数是用来对传入的信息做相关预处理的，以便于后续的矩阵操作。

3. main 函数板块

功能简介：本板块的功能为通过函数 w.show()启动类 MAP 来定义用户界面主窗口 w，启动用户主界面，进行可视化操作。

3 软件开发与测试

3.1 软件开发

硬件环境：

- 处理器：AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz
- 机带 RAM：16.0GB(15.4GB 可用)
- 系统类型：64 位操作系统，基于 x64 的处理器

开发环境：MSVC 2019 C++11，编译模式为 X86。 部分代码在 gcc， gdb 环境调试完成。

3.2 软件测试

1. 打开程序，显示地图，显示结果如图 3.1 所示。

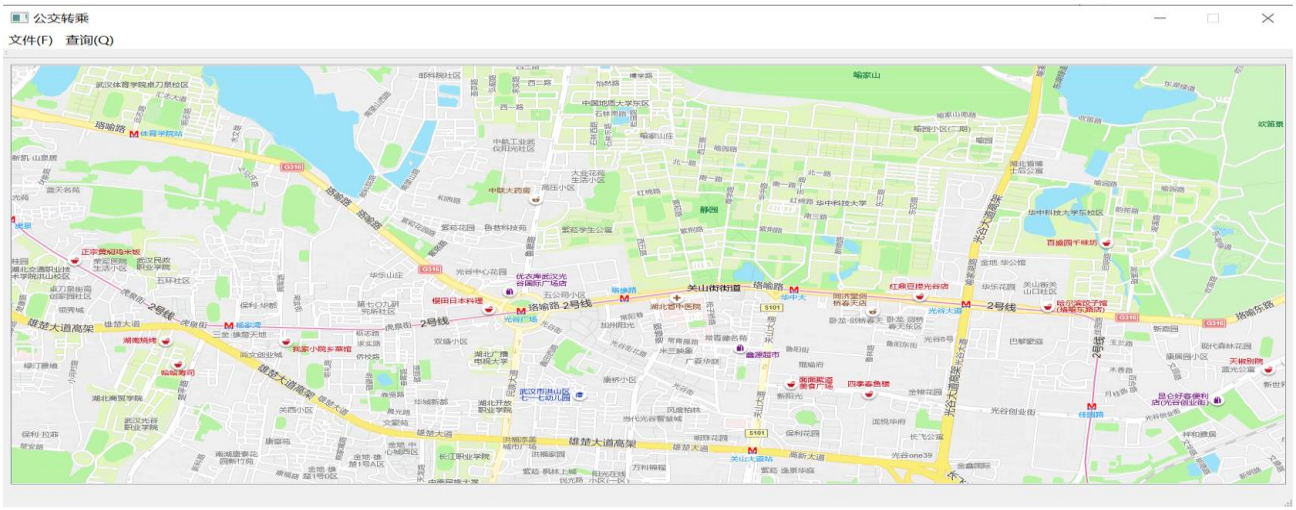


图 3. 1

2. 读入相关文件，界面如图 3.2 所示，显示结果如图 3.3 所示。

图 3. 2

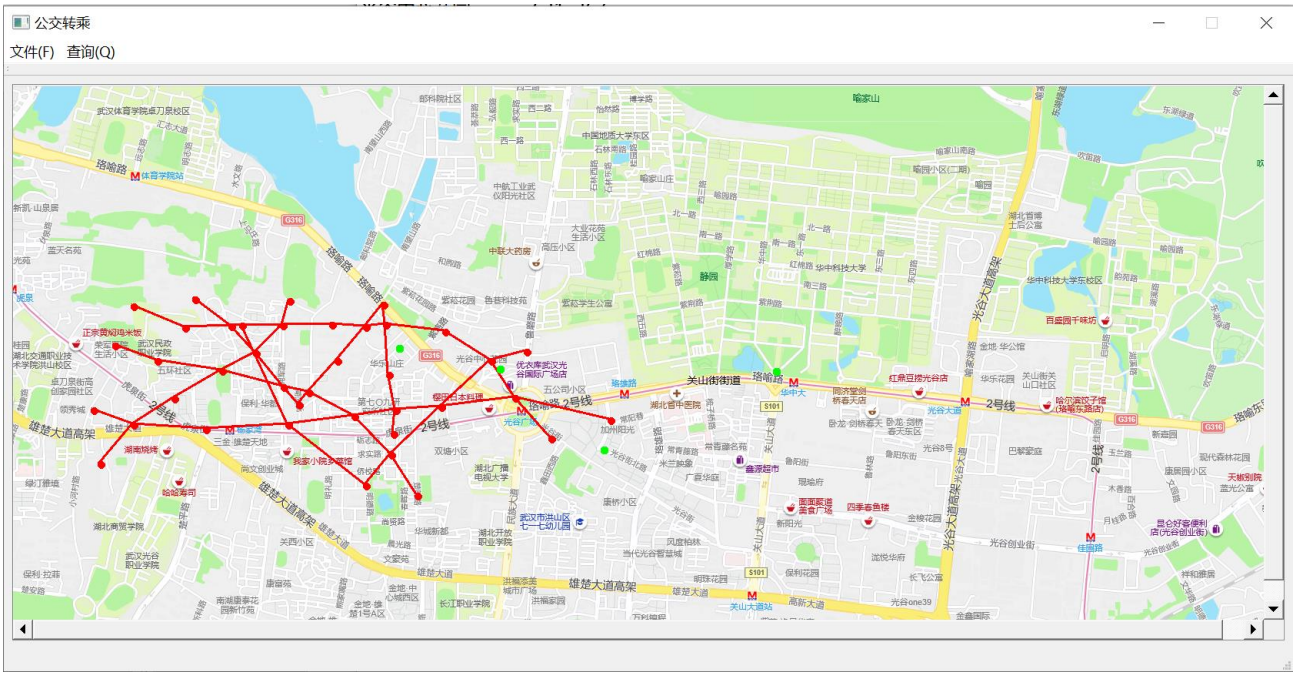


图 3. 3

其中线路和站点由红色标注，地名文件内容由绿色标注。

3. 鼠标选择起点和终点，其中起点用棕色标注，使用鼠标左键点击获取，终点用黄色标注，使用鼠标右键点击获取，其运行结果如图 3.4 所示。

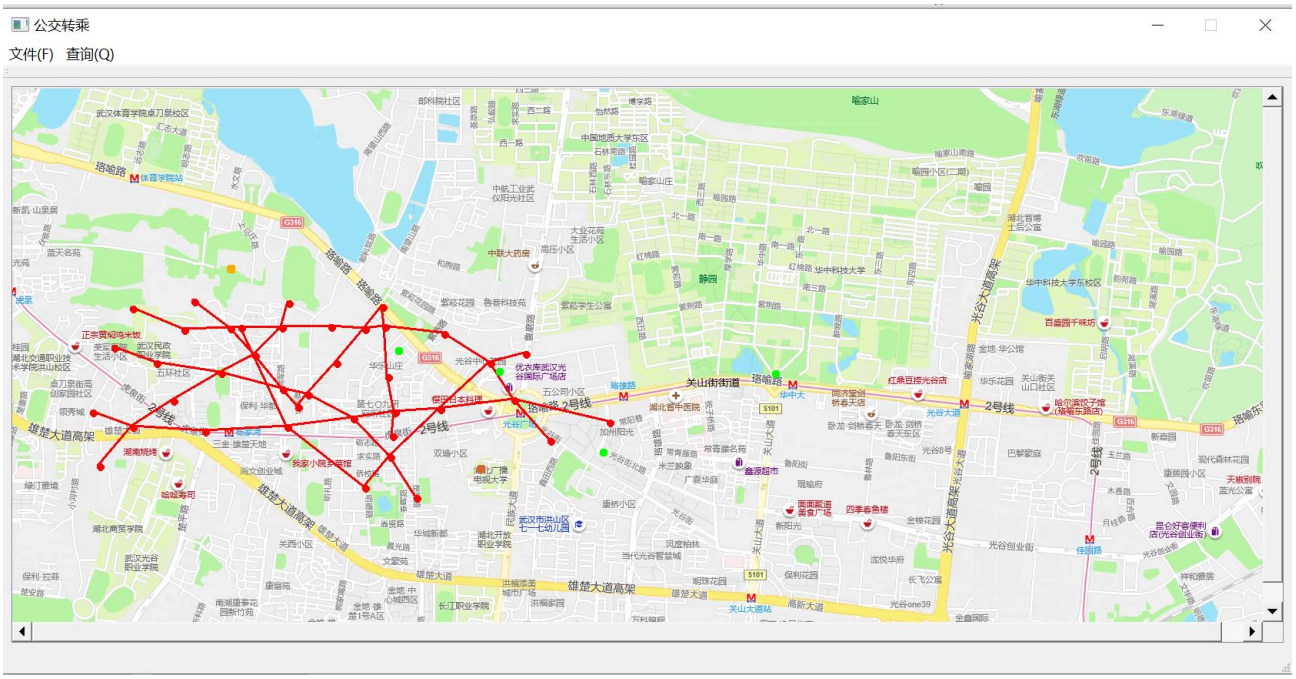


图 3. 4

4. 最少转乘查询结果如图 3.5 所示，查询得到的路径为其中的蓝色路径。

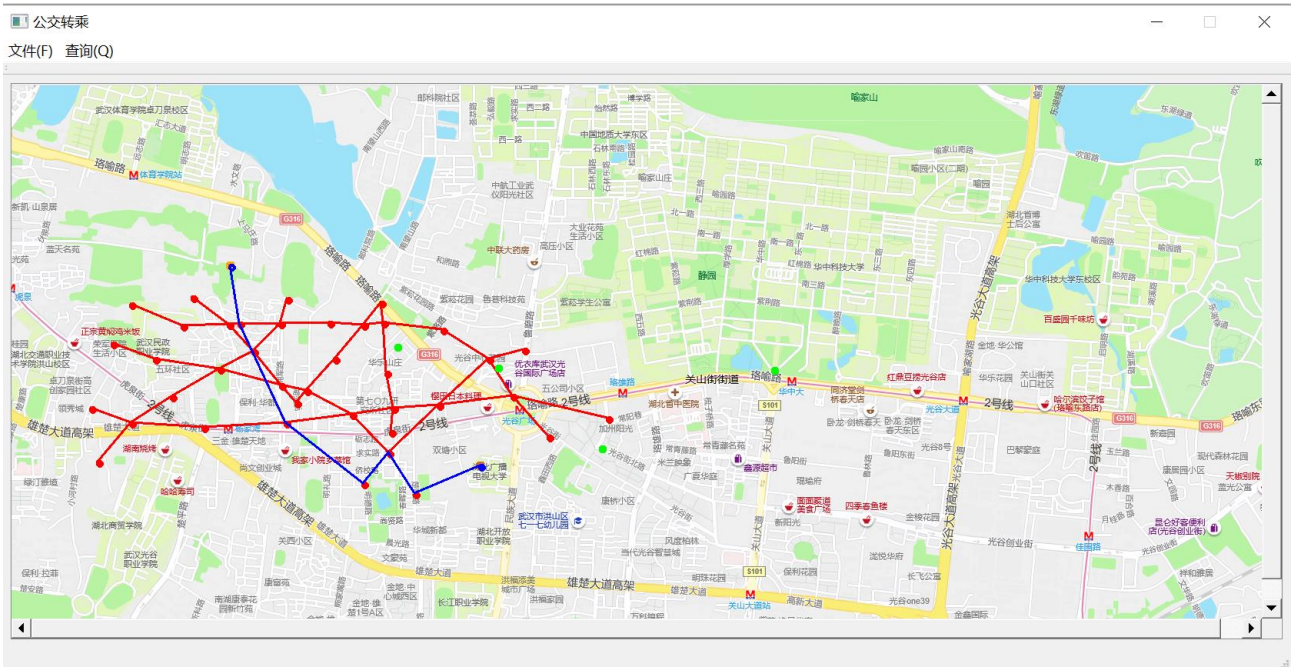


图 3. 5

5. 最短距离的查询结果如图 3.6 蓝色路径所示。

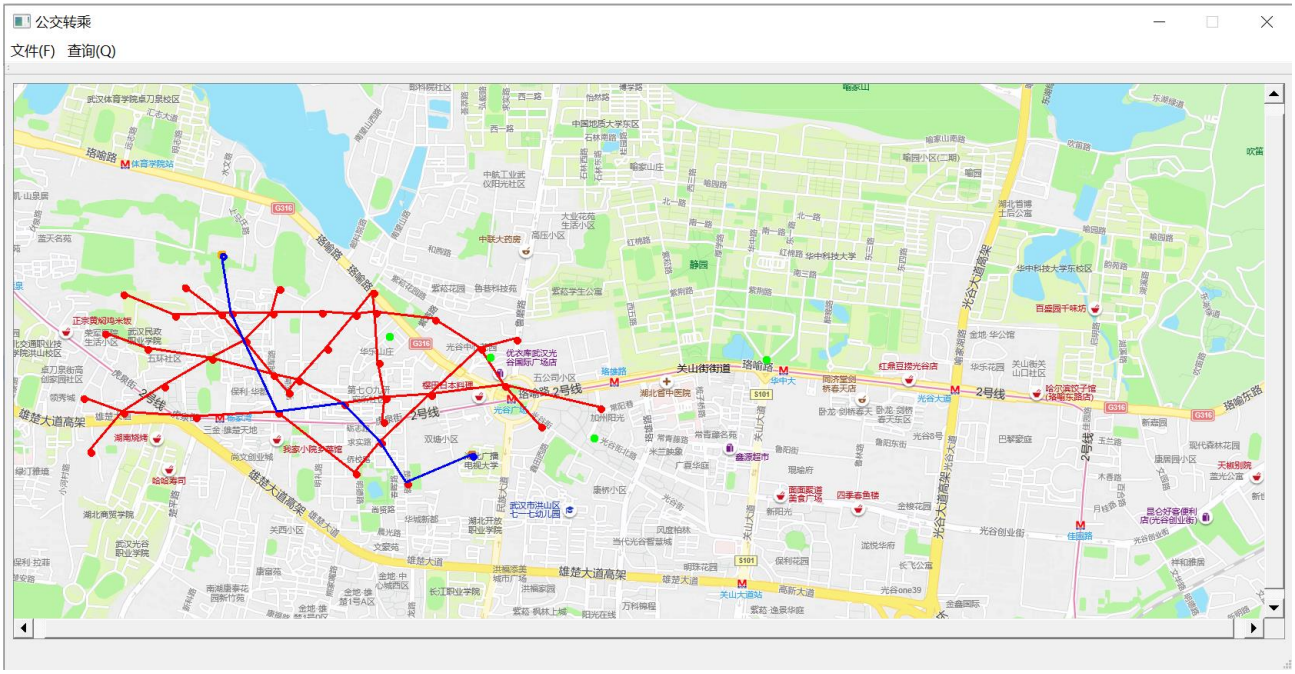


图 3. 6

6. 模糊匹配界面测试

在点击模糊匹配选项之后，出现了模糊匹配界面，如图 3.7 所示。

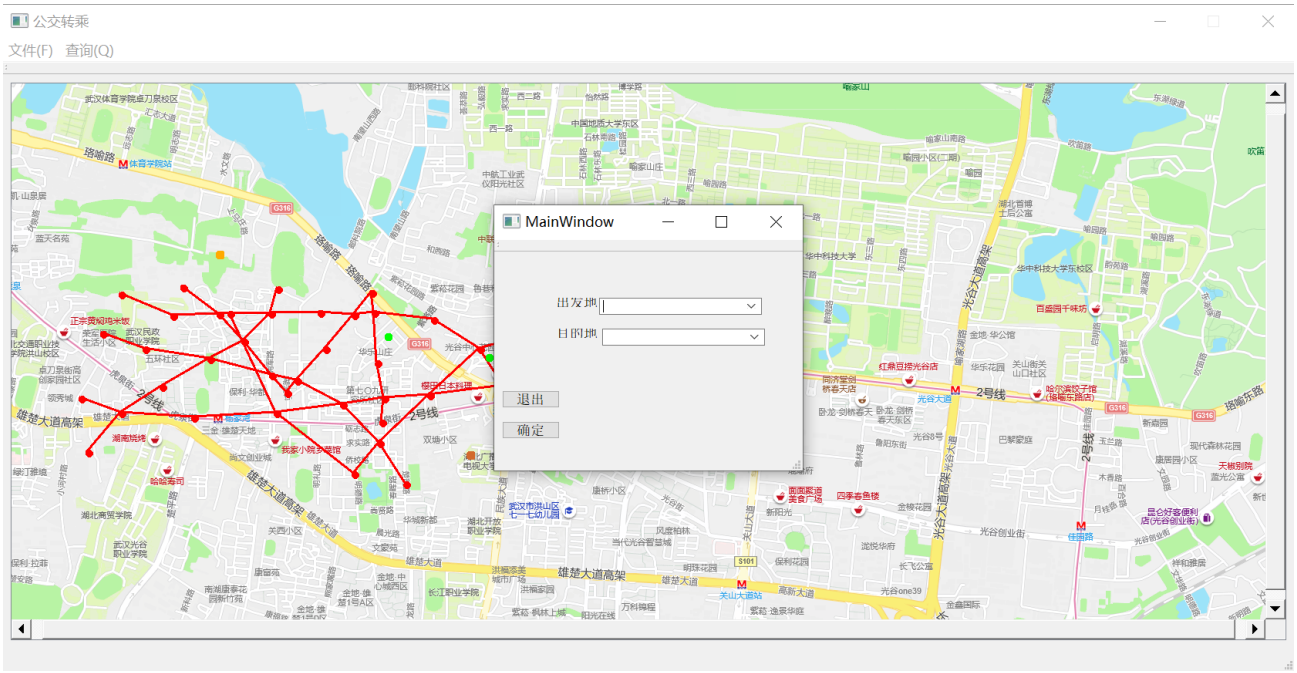


图 3. 7

输入地名效果如图 3.8 所示。

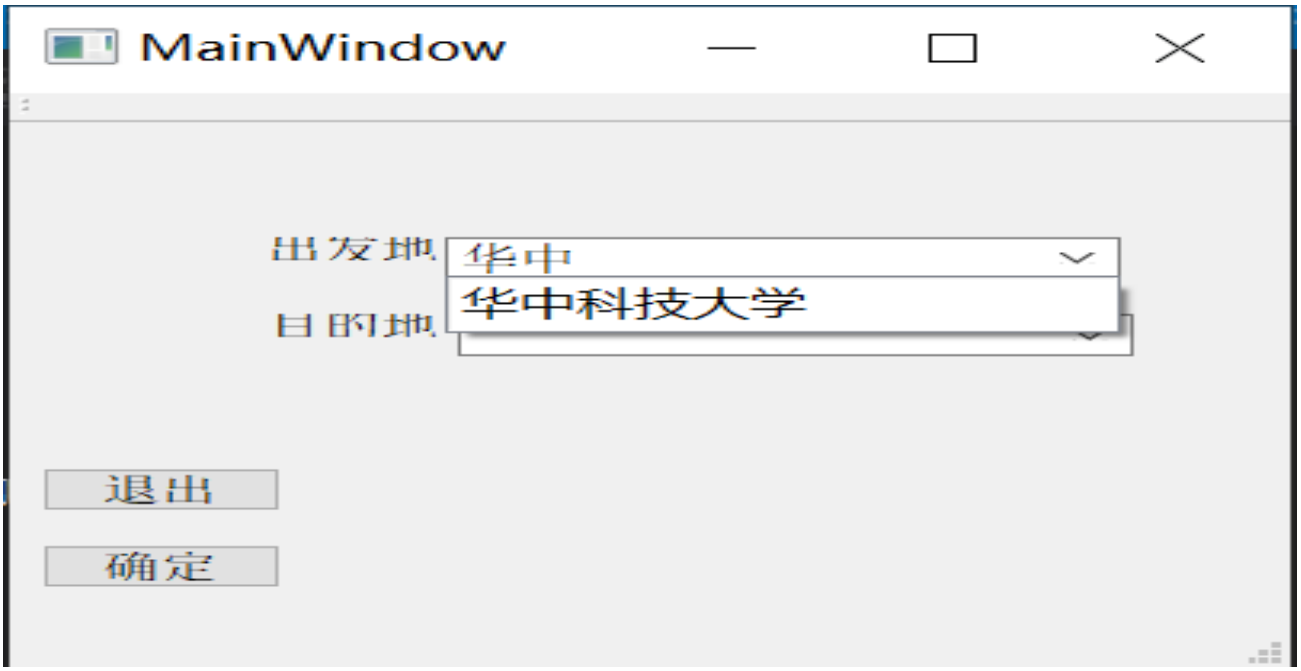


图 3. 8

7. 在各个界面均有相应的关闭与确定按钮，此处不再测试。

4 特点与不足

4.1 技术特点

本次实验结合 QT 与 C++，实现了一个简易的公交转乘系统。用户通过输入文件路径可以显示当前的站点、线路以及已有的地点名，在鼠标移动到站点上的时候可以显示这是几号站点，当鼠标移动到绿色点时可以显示当前的地名，可以使用鼠标左键选择出发点并使用鼠标右键选择目的地，并通过相关操作得到两个地点间的最少转乘或最短距离。同时也实现了模糊匹配的界面，用户可以通过输入地名来选择出发点和目的地。

4.2 不足和改进的建议

本次实验完成了通过鼠标获取出发点和目的地并进行查询的功能，但是在模糊匹配部分，仅仅做出了模糊匹配的界面，并没有使其具备查询最少转乘或最短距离的功能，这是有所欠缺的地方。

5 过程和体会

5.1 遇到的主要问题和解决方法

1. 本次实验的工程量很大，不管是最开始的逻辑层实现还是后来使用 Qt 做出图形化界面，都要耗费很大的力气，我也想了很久也没有很好的思路，后来在参考了马光志老师的教学视频后顺利的做出了使用鼠标选择出发点和目的地并进行查询的功能。

2. 在设计模糊匹配的界面的时候，无法正确读入汉字，在同学的帮助下，修改 txt 文件的编码格式即可正确读入。

3. 在读入文件时，对 org 的 name 成员赋值错误，最开始是采用类似于移动拷贝的方法将读入数据进行拷贝，但是却无法正常读入，在调试的时候发现 vs2019 会对在循环中的局部指针变量指向同一块地址，导致移动拷贝出错，最终在成员函数中为其分配了一块新的空间之后使用深拷贝，解决了这一问题。

4. 本次实验使用 QLineEdit 与 QCombox 组合实现模糊匹配，但是在建立 QStringList 类型的 qstr 时，需要使用强制类型转换，否则无法正常显示汉字。

5.2 课程设计的体会

这次实验给我最大的体会是 C++功能的强大，我们可以凭借书本上学习的知识完成公交路线导航这一具有实际作用的应用。这充分体现了计算机专业是一门理论与实践相结合的课程。同时，自学 QT 的经验也告诉我，在今后的学习中，有许多东西都需要自己去理解和掌握，要养成自学的习惯，提升自己的能力。

6 源码和说明

6.1 文件清单及其功能说明

在所提交文件中包含源文件和相关测试文件。

6.2 用户使用说明书

用户启动程序后，首先点击“文件”菜单的“读入数据”按钮，在弹出的窗口中点击“浏览...”按钮读入站点文件、路线文件以及地名文件。然后程序会在地图上显示站点、路线以及地名。用户可将鼠标放在站点上，会出现提示信息。然后用户通过鼠标左键选择起点，鼠标右键选择终点。用户可在“查询”菜单中选择“最少转乘”或“最短距离”按钮。选择后会将路线显示在地图上。当使用完毕后，点击“退出”按钮退出程序。此外，用户也可以点击“文件”菜单的“模糊匹配”按钮进行输入地名选择出发点和目的地，但是不能通过这种方法进行查询。

6.3 源代码

logiclay.h:

```
#pragma once
#include<string.h>
class STOP {                                    //描述一个公交站点
    int numb;                                   //所有公交站点编号
    int x, y;                                   //公交站点坐标
public:
    STOP(int n = 0, int x = 0, int y = 0);
    virtual int& X();
    virtual int& Y();
    virtual int& N();
};
class LINE {                                    //描述一条公交线路
    const int numb;                             //公交线路编号从 1 开始
    int* const stop;                            //公交线路上所有站点编号
    const int nofs;                             //公交线路上站点数量
public:
```

```
    LINE(int numb = 0, int nofs = 0, int* stop = nullptr);
    LINE(const LINE& r);
    LINE(LINE&& r)noexcept;
    LINE& operator=(const LINE& r);
    LINE& operator=(LINE&& r)noexcept;
    virtual int has(int s)const;                    //若包含站点编号 s，则返回线路中的站次序
号；-1 表示未包含
    virtual int cross(const LINE& b)const;          //若两条公交线路相交，则返回 1
    virtual operator int()const;                    //取公交线路编号
    virtual int NOFS()const;                        //取公交线路的站点数量
    virtual double dist(int d, int a)const;         //线路从站次 d 到站次 a 的距离
    virtual int& operator[](int x);                 //取线路某个站次的站点编号
    virtual ~LINE()noexcept;
};
class TRAN {                                        //从线路 from 经站点编号 stop 转至线
路 to
    int from;                                       //现乘的公交线路号
    int to;                                         //需要转乘的公交线路号
    int stop;                                       //由 stops.txt 定义的站点编号
public:
    TRAN(int from = 0, int to = 0, int stop = 0);
    int operator==(const TRAN& t)const;
    virtual int& F();                               //现乘的公交线路号
    virtual int& T();                               //需要转乘的公交线路号
    virtual int& S();                               //转乘点的站点编号
};
class ROUTE {                                       //一个转乘路径
    TRAN* const tran;                               //转乘路径上的所有转乘站点
    const int noft;                                 //转乘路径上的转乘次数
public:
    ROUTE(TRAN* tran = nullptr, int noft = 0);
    ROUTE(const TRAN& t);
    ROUTE(const ROUTE& r);
    ROUTE(ROUTE&& r)noexcept;
    virtual operator int()const;                    //得到转乘次数
    virtual int operator==(const ROUTE& r)const;
```

```
    virtual ROUTE operator *()const;                    //约简：去掉重复的公交转乘线路
    virtual TRAN& operator[](int);//一条路径上的所有转乘站点
    virtual ROUTE operator+(const ROUTE& r)const;// 转乘路径连接
    virtual ROUTE& operator=(const ROUTE& r);
    virtual ROUTE& operator=(ROUTE&& r)noexcept;
    virtual ROUTE& operator+=(const ROUTE& r);
    virtual ~ROUTE()noexcept;
    virtual int print()const;
    //打印转乘路径
};
class NODE {//矩阵元素：记载的转乘次数和线路
    ROUTE* const p;//矩阵 R*c 个元素的转乘路径方案
    int n;//矩阵 r*c 个元素记载的转乘路径方案数
public:
    NODE(ROUTE* p, int n);
    NODE(int n = 0);
    NODE(const NODE& n);
    NODE(NODE&& n)noexcept;
    virtual NODE operator*()const;//矩阵元素约简：去掉转乘中的环
    virtual NODE operator+(const ROUTE& n)const;//元素增加路径
    virtual NODE operator+(const NODE& n)const;//元素路径增加
    virtual NODE operator*(const NODE& n)const;//元素路径转乘连接
    virtual NODE& operator=(const NODE& n);
    virtual NODE& operator+=(const NODE& n);
    virtual NODE& operator+=(const ROUTE& n);
    virtual NODE& operator*=(const NODE& n);
    virtual NODE& operator=(NODE&& n)noexcept;
    virtual ROUTE& operator[](int x);//获得第 X 个转乘路径
    virtual operator int& ();//返回可转乘路径数 n
    virtual ~NODE()noexcept;
    virtual void print()const;//打印转乘矩阵元素

};
class organization {
    const char *name;
    int x, y;
```

```
public:
    organization(const char* name=nullptr, int x=0, int y=0);
    virtual int& X();
    virtual int& Y();
    virtual const char* NAME();
    void NAME(const char* &ch)
    {
        name =new char[strlen(ch)+1];
        strcpy((char*&)name, ch);
        ch = nullptr;
    }
};
class TMAP {//所有公交转乘元素的“闭包”矩阵
    NODE* const p;
    const int r, c;//行数和列数
public:
    TMAP(int r = 0, int c = 0);
    TMAP(const TMAP& a);
    TMAP(TMAP&& a)noexcept;
    virtual ~TMAP();
    virtual int notZero()const;//若出现不可到达站点，就返回 0
    //以下函数返回最少转乘数的路径数，起点站次为 b，终点站次为 e,r 存在 10 条路径
    virtual int miniTran(int b, int e, int& noft, ROUTE(&r)[100])const;
    //以下函数返回最短距离的路径数，起点站次为 b,终点站次为 e,,r 存在十条路径
    virtual int miniDist(int b, int e, double& dist, ROUTE(&r)[100])const;
    static double getDist(int b, int e, ROUTE& r);
    virtual NODE* operator[](int r);//得到矩阵某行 r 元素 的首地址
    virtual int& operator()(int r, int e);
    virtual TMAP operator*(const TMAP& a)const;//“闭包”运算：乘法
    virtual TMAP operator+(const TMAP& a)const;//"闭包“运算：加法
    virtual TMAP& operator=(const TMAP& a);
    virtual TMAP& operator= (TMAP&& a);
    virtual TMAP& operator+=(const TMAP& a);
    virtual TMAP& operator*=(const TMAP& a);
    virtual TMAP& operator()(int r, int c, const ROUTE& a);//将路径 a 加入矩阵元素中
    virtual void print()const;//打印转置矩阵
```

```
};
struct GIS {//描述地理信息系统的类
    static STOP* st;//所有公交站点数
    static organization* org;
    static LINE* ls;//所有公交线路
    static int ns, nl,no;//公交站数 ns,公交线路数,org 站点数
    static TMAP raw, tra;//原始转乘矩阵 raw,“闭包”转乘矩阵
    static int obs;//GIS 的对象数量
    static int numoforg;
    int start, end;
    bool flag = false;
public:
    GIS();
    GIS(const char* flstop, const char* flline,const char* florg=nullptr);//用站点及线路文件加载地图
    int miniTran(int fx, int fy, int tx, int ty, int& f, int& t, int& n, ROUTE(&r)[100]);
    int miniDist(int fx, int fy, int tx, int ty, int& f, int& t, double& d, ROUTE(&r)[100]);
    ~GIS();
};
extern GIS* gis;
```

logiclay.cpp:

```
#define  _CRT_SECURE_NO_WARNINGS
#include "logiclayer.h"
#include <math.h>
#include <stdio.h>


STOP* GIS::st = 0;        //公交站
LINE* GIS::ls = 0;        //公交线路
int   GIS::ns = 0;         //公交站数
int   GIS::nl = 0;         //公交线路数
int GIS::no = 0;          //org 站点数目
int   GIS::obs = 0;        //GIS 对象数量
int GIS::numoforg = 0;
organization* GIS::org = 0;
TMAP GIS::raw;              //原始转移图
TMAP GIS::tra;              //闭包转移图
```

```
STOP::STOP(int n, int x, int y) : numb(n), x(x), y(y) { }
int& STOP::X() { return x; }
int& STOP::Y() { return y; }
int& STOP::N() { return numb; }
//================================LINE=========================================
========
LINE::LINE(int n, int c, int* stop) : numb(n), stop(c ? new int[c] : nullptr), nofs(stop ? c : 0) {
    if (LINE::stop == nullptr && nofs != 0) throw "Memory allocation for bus line error!";
    for (int x = 0; x < nofs; x++)
        LINE::stop[x] = stop[x];
}
LINE::LINE(const LINE& r) : numb(r.numb), stop(r.nofs ? new int[r.nofs] : nullptr), nofs(stop ? r.nofs : 0) {
    if (stop == nullptr && r.nofs != 0) throw "Memory allocation for bus line error!";
    for (int x = 0; x < nofs; x++)
        stop[x] = r.stop[x];
}
LINE::LINE(LINE&& r) noexcept : numb(r.numb), stop(r.stop), nofs(r.nofs) {
    (int&)(r.numb) = (int&)(r.nofs) = 0;
    (int*&)(r.stop) = nullptr;
}
LINE& LINE::operator=(const LINE& r) {
    if (this == &r) return *this;
    if (stop) { delete[] stop; }
    (int&)numb = r.numb;
    (int*&)stop = new int[r.nofs];
    if (stop == nullptr) throw "Memory allocation for operator= of bus line error!";
    (int&)nofs = stop ? r.nofs : 0;
    for (int x = 0; x < nofs; x++) stop[x] = r.stop[x];
    return *this;
}
LINE& LINE::operator=(LINE&& r)noexcept {
    if (this == &r) return *this;
    if (stop) { delete[] stop; }
    (int&)numb = r.numb;
    (int*&)stop = r.stop;
```

```
    (int&)nofs = r.nofs;
    (int&)(r.numb) = (int&)(r.nofs) = 0;
    (int*&)(r.stop) = nullptr;
    return *this;
}
int LINE::has(int s)const {
    for (int x = 0; x < nofs; x++)
        if (stop[x] == s) return x;
    return -1;
}
int LINE::cross(const LINE& b)const {    //线路相交则返回 1
    if (this == &b) return 0;
    for (int x = 0; x < nofs; x++)
        if (b.has(stop[x]) != -1) return 1;
    return 0;
}
LINE::operator int() const { return numb; }          //取线路编号
int LINE::NOFS()const { return nofs; }               //取站点数目
double LINE::dist(int b, int e)const { //线路从站点 b(数组下标)到站点 e(数组下标)的距离
    double d;
    int bi, ei; //两个站点的开始及结束序号
    int x1, y1, x2, y2, w;
    if ((bi = has(b)) == -1) throw "Wrong stop number while calculating distance!";
    if ((ei = has(e)) == -1) throw "Wrong stop number while calculating distance!";
    if (bi > ei) { b = ei; ei = bi; bi = b; }
    d = 0;
    for (b = bi; b < ei; b++) {
        x1 = GIS::st[w = stop[b]].X();   //由当前站点编号得到其坐标位置 x
        y1 = GIS::st[w].Y();                  //由当前站点编号得到其坐标位置 y
        x2 = GIS::st[e = stop[b + 1]].X() - x1;        //到下一站的 x 轴之差
        y2 = GIS::st[e].Y() - y1;                        //到下一站的 y 轴之差
        d += sqrt(double(x2 * x2 + y2 * y2));          //到下一站的距离
    }
    return d;
}
int& LINE::operator[](int x) { return stop[x]; }      //第 x 站的站点编号
```

```
LINE::~LINE()noexcept { if (stop) { delete[] stop; (int*&)stop = 0; (int&)numb = (int&)nofs = 0; } }
//=======================================TRAN=================================
==============
TRAN::TRAN(int from, int to, int stop) : from(from), to(to), stop(stop) { }
int TRAN::operator==(const TRAN& t)const {
	return from == t.from && to == t.to && stop == t.stop;
}
int& TRAN::F() { return from; }
int& TRAN::T() { return to; }
int& TRAN::S() { return stop; }
//=======================================ROUTE================================
=============
ROUTE::ROUTE(TRAN* tran, int noft) : tran(noft ? new TRAN[noft] : nullptr), noft(tran ? noft : 0) {
	if (ROUTE::tran == nullptr && noft != 0) throw "memory allocation for ROUTE construction error!";
	for (int x = 0; x < noft; x++) ROUTE::tran[x] = tran[x];
}
ROUTE::ROUTE(const TRAN& t) : tran(new TRAN[1]), noft(tran ? 1 : 0) {
	if (tran == nullptr) throw "memory allocation for ROUTE construction error!";
	if (tran) *tran = t;
}
ROUTE::ROUTE(const ROUTE& r) : tran(r.noft ? new TRAN[r.noft] : nullptr), noft(tran ? r.noft : 0) {
	if (tran == nullptr && r.noft != 0) throw "memory allocation for ROUTE construction error!";
	for (int x = 0; x < noft; x++) tran[x] = r.tran[x];
}
ROUTE::ROUTE(ROUTE&& r) noexcept : tran(r.tran), noft(r.noft) {
	(TRAN*&)(r.tran) = nullptr;
	(int&)(r.noft) = 0;
}
ROUTE::~ROUTE() noexcept {
	if (tran) {
		delete[]tran;
		(TRAN*&)tran = nullptr;
		*(int*)&noft = 0;
	}
}
int ROUTE::print()const {
```

```
    for (int x = 0; x < noft; x++)
        if (tran[x].S() == -1) printf("\tstay on line %d and go directly\n", tran[x].F() + 1);
        else       printf("\tfrom line %d to line %d via %d\n", tran[x].F() + 1, tran[x].T() + 1, tran[x].S() + 1);
    return noft;
}
ROUTE::operator int()const { return noft; }
ROUTE ROUTE::operator *()const { //去重复转乘
    int nn = noft;
    TRAN* t = new TRAN[noft];
    if (t == nullptr) throw "memory allocation for ROUTE::operator*() error!";
    for (int x = 0; x < nn; x++) t[x] = tran[x];
    for (int x = 0; x < nn - 1; x++)
        for (int y = x + 1; y < nn; y++) {
            if (t[x].S() == t[y].S() && t[x].F() == t[y].T()) { //循环了
                for (int m = x, n = y + 1; n < nn; n++, m++) t[m] = t[n];
                nn -= (y + 1 - x);
                y = x;
            }
        }
    ROUTE r(t, nn);
    delete[]t;
    return r;
}
int ROUTE::operator==(const ROUTE& r)const {
    int m = 1;
    if (noft != r.noft) return 0;
    for (int x = 0; x < noft; x++) {
        if (tran[x] == r.tran[x]) continue;
        m = 0;
        break;
    }
    return m;
}
TRAN& ROUTE::operator[](int x) { //一条线路上所有转乘
    if (x<0 || x>noft) throw "Subscript bound error!";
    return tran[x];
```

```
}
ROUTE    ROUTE::operator+(const ROUTE& r)const {
    int x, y;
    ROUTE s;
    if (noft == 0) return *this;
    if (r.noft == 0) return r;
    //两个 ROUTE 均不为空时
    if (tran[noft - 1].T() != r.tran[0].F())
        throw "Route can nott be concated!";
    try {
        (TRAN*&)(s.tran) = new TRAN[noft + r.noft];
    }
    catch (...) {
        throw "Memory allocation for ROUTE construction error!";
    }
    (int&)(s.noft) = s.tran ? noft + r.noft : 0;
    for (x = 0; x < noft; x++) s.tran[x] = tran[x];
    for (y = 0; y < r.noft; y++) s.tran[x++] = r.tran[y];
    return *s;
}
ROUTE& ROUTE::operator=(const ROUTE& r) {
    if (this == &r) return *this;
    if (tran) delete[]tran;
    try {
        (TRAN*&)tran = new TRAN[r.noft];
    }
    catch (...) {
        throw "Memory allocation for ROUTE construction error!";
    }
    (int&)noft = tran ? r.noft : 0;
    for (int x = 0; x < noft; x++) tran[x] = r.tran[x];
    return *this;
}
ROUTE& ROUTE::operator=(ROUTE&& r) noexcept {
    if (this == &r) return *this;
    if (tran) delete[]tran;
```

```
    (TRAN*&)tran = r.tran;
    (int&)noft = r.noft;
    (TRAN*&)(r.tran) = nullptr;
    (int&)(r.noft) = 0;
    return *this;
}
ROUTE& ROUTE::operator+=(const ROUTE& r) { return *this = *this + r; }

//=====================================NODE=====================================
===
NODE::NODE(ROUTE* p, int n) :p(n ? new ROUTE[n] : nullptr), n(p ? n : 0) {
    if (NODE::p == nullptr && n != 0) throw "memory allocation for ROUTE construction error!";
    for (int x = 0; x < n; x++) NODE::p[x] = p[x];
}
NODE::NODE(int n) : p(n ? new ROUTE[n] : nullptr), n(p ? n : 0) {
    if (p == nullptr && n != 0) throw "memory allocation for NODE construction error!";
}
NODE::NODE(const NODE& n) : p(n.n ? new ROUTE[n.n] : nullptr), n(p ? n.n : 0) {
    if (p == nullptr && n.n != 0) throw "memory allocation for NODE construction error!";
    for (int x = 0; x < NODE::n; x++) p[x] = n.p[x];
}
NODE::NODE(NODE&& n)noexcept : p(n.p), n(n.n) {
    (ROUTE*&)(n.p) = nullptr;
    (int&)n.n = 0;
}
NODE    NODE::operator*()const {
    int n = NODE::n;
    if (n == 0) return *this;
    ROUTE* t = new ROUTE[n];
    if (t == nullptr) throw "memory allocation for NODE::operator*() error!";
    for (int x = 0; x < n; x++) t[x] = p[x];
    for (int x = 0; x < n - 1; x++)
        for (int y = x + 1; y < n; y++) {
            if (t[x] == t[y]) {
                for (int m = x + 1, n = y + 1; n < n - 1; n++, m++)
                    t[m] = t[n];
```

```
                n -= (y - x);
                y = x;
            }
        }
    NODE r(t, n);
    try {
        if (t != nullptr)delete[]t;
        t = nullptr;
    }
    catch (...) {
        throw "initializing failed！ ";
    }
    return r;
}
NODE    NODE::operator+(const ROUTE& n)const {
    NODE r(NODE::n + 1);
    for (int x = 0; x < NODE::n; x++) r.p[x] = *p[x];
    r.p[NODE::n] = n;
    return *r;
}
NODE    NODE::operator+(const NODE& n)const {
    if (NODE::n == 0) return n;
    if (n.n == 0) return *this;
    NODE r(NODE::n + n.n);
    for (int x = 0; x < NODE::n; x++) r.p[x] = *p[x];
    for (int x = 0; x < n.n; x++) r.p[x + NODE::n] = *n.p[x];
    return *r;
}
NODE    NODE::operator*(const NODE& n)const {
    if (NODE::n == 0) return *this;
    if (n.n == 0) return n;
    NODE r(NODE::n * n.n);
    int m, f, h, k = 0;
    for (int x = 0; x < NODE::n; x++)      //当前节点:x=4,y=0,k=4 时异常
        for (int y = 0; y < n.n; y++) {
            if (p[x][-1 + p[x]].T() != n.p[y][0].F()) throw "Can not tansship from buses!";
```

```
                try {
                    r.p[k] = p[x] + n.p[y];    //Route 的连接运算
                }
                catch (const char* e) {
                    const char* p = e;
                }
                k++;
            }
    return *r;
}
NODE& NODE::operator=(const NODE& n) {
    if (this == &n) return *this;
    if (p) delete[]p;
    (ROUTE*&)p = new ROUTE[n.n];
    if (p == nullptr) throw "Memory allocation for NODE construction error!";
    (int&)(NODE::n) = p ? n.n : 0;
    for (int x = 0; x < n.n; x++) p[x] = n.p[x];
    return *this;
}
NODE& NODE::operator=(NODE&& n)noexcept {
    if (this == &n) return *this;
    if (p) delete[]p;
    (ROUTE*&)p = n.p;
    (int&)(NODE::n) = n.n;
    (ROUTE*&)(n.p) = nullptr;
    (int&)(n.n) = 0;
    return *this;
}
NODE& NODE::operator+=(const ROUTE& n) {
    return *this = *this + n;
}
NODE& NODE::operator+=(const NODE& n) {
    return *this = *this + n;
}
NODE& NODE::operator*=(const NODE& n) {
    return *this = *this * n;
```

```
}
ROUTE& NODE::operator [](int x) {
	if (x < 0 || x >= n) throw "Subscription x of NODE::operator [](int x) is wrong!";
	return p[x];
}
NODE::operator int& () { return n; }
NODE::~NODE()noexcept {
	if (p) {
		delete[]p;
		(ROUTE*&)p = nullptr;
		(int&)n = 0;
	}
}
void NODE::print()const {
	for (int m = 0; m < n; m++) {
		p[m].print();
	}
}
//==================================TMAP=======================================
====
TMAP::TMAP(int r, int c) : p((r != 0 && c != 0) ? new NODE[r * c] : nullptr), r(p ? r : 0), c(p ? c : 0) {
	if (TMAP::p == nullptr && r != 0 && c != 0) throw "Memory allocation for TMAP construction error!";
}
TMAP::TMAP(const TMAP& a) : p((a.r != 0 && a.c != 0) ? new NODE[a.r * a.c] : nullptr), r(p ? a.r : 0), c(p ?
a.c : 0) {
	if (p == nullptr && a.r != 0 && a.c != 0) throw "Memory allocation for TMAP construction error!";
	for (int k = r * c - 1; k >= 0; k--) p[k] = a.p[k];
}
TMAP::TMAP(TMAP&& a)noexcept : p(a.p), r(a.r), c(a.c) {
	(NODE*&)(a.p) = nullptr;
	(int&)(a.r) = (int&)(a.c) = 0;
}
TMAP::~TMAP() {
	if (p) {
		delete[] p;
		(NODE*&)p = nullptr;
```

```
            (int&)r = (int&)c = 0;
        }
}
int TMAP::notZero()const {
        for (int x = r * c - 1; x >= 0; x--) if (p[x].operator int& () == 0) return 0;
        return 1;
}
int& TMAP::operator()(int x, int y) {
        if (x < 0 || x >= r) throw "Subscript bound error!";
        if (y < 0 || y >= c) throw "Subscript bound error!";
        return p[x * c + y];
}
NODE* TMAP::operator[](int r) {
        if (r < 0 || r >= TMAP::r) throw "Subscript bound error!";
        return p + r * c;
}
TMAP& TMAP::operator=(const TMAP& a) {
        if (this == &a) return *this;
        if (p) delete[]p;
        (NODE*&)p = new NODE[a.r * a.c];
        if (p == nullptr) throw "Memory allocation for TMAP assignment error!";
        (int&)r = a.r;
        (int&)c = a.c;
        for (int k = r * c - 1; k >= 0; k--)
            p[k] = a.p[k];
        return *this;
}
TMAP& TMAP::operator=(TMAP&& a) {
        if (this == &a) return *this;
        if (p) delete[]p;
        (NODE*&)p = a.p;
        (int&)r = a.r;
        (int&)c = a.c;
        (NODE*&)(a.p) = nullptr;
        (int&)(a.r) = (int&)(a.c) = 0;
        return *this;
```

```
}
TMAP TMAP::operator*(const TMAP& a)const {
    if (c != a.r) throw "TMAP can nott mutiply!";
    int t, m, u, v, w, x, y, z;
    TMAP s(r, a.c);                          //每个节点皆为空线路
    for (int h = 0; h < r; h++)
        for (int j = a.c - 1; j >= 0; j--) {
            if (h == j) continue;
            t = h * s.c + j;
            //s.p[t] = NODE();          //原有路线数:方阵
            for (int k = 0; k < c; k++)
                if (k != h && k != j) //新增路线数
                    s.p[t] += p[h * c + k] * a.p[k * a.c + j];
        }
    return s;
}
TMAP TMAP::operator+(const TMAP& a)const {
    if (r != a.r && c != a.c) throw "TMAP can nott add!";
    TMAP s(*this);
    for (int h = r * c - 1; h >= 0; h--) s.p[h] += a.p[h];
    return s;
}
TMAP& TMAP::operator+=(const TMAP& a) { return *this = *this + a; }
TMAP& TMAP::operator*=(const TMAP& a) { return *this = *this * a; }
TMAP& TMAP::operator()(int ro, int co, const ROUTE& a) {
    p[ro * c + co] += a;
    return *this;
}
//根据起点站 s 和终点站 t 找到最少转乘次数 noft 的若干线路 r,返回线路数
int TMAP::miniTran(int s, int t, int& notf, ROUTE(&r)[100])const {
    int k, u, v, w, x, y, z;//z:返回实际最少转乘线路数
    int b = 0, e = 0;            //包含起点 s 的起始线路数 b(线路存放在 bls),包含终点 t 的起始线路数 e(线路存放在 els)
    int nott[100]{};             //对应规划线路 r 的转乘次数
    int bls[20], els[20];     //bls:包含起点 s 的起始线路
    NODE rou;
```

```
    for (z = 0; z < GIS::nl; z++) { //寻找包含起点或终点相关公交线路下标
        if (GIS::ls[z].has(s) != -1) if (b < 20) bls[b++] = z;
        if (GIS::ls[z].has(t) != -1) if (e < 20) els[e++] = z;
    }
    for (x = z = 0; x < b; x++)
        for (y = 0; y < e; y++) {
            rou = GIS::tra[bls[x]][els[y]]; //得到两线路的所有转乘线路
            w = GIS::tra(bls[x], els[y]);
            if (w == 0) continue;     //转乘线路数==0
            for (v = 0; v < w; v++) {
                u = rou[v];               //线路得到转乘次数
                if (z == 0 || u < nott[0]) {//若 nott 为空，或转乘次数比 nott[0]小
                    nott[0] = u;
                    r[0] = rou[v];
                    z = 1;
                }
                if (u == nott[0]) { //和已有线路转乘次数相同时，则插入
                    if (z == 100) return z;
                    nott[z] = u;
                    r[z] = rou[v];
                    z++;
                }
            }
        }
    notf = nott[0]; //nott[0]到的 nott[z-1]的转乘次数都相同
    return z;           //返回最少转乘线路数
}
//起站 b,终站 e,使用路线 r 的距离
double TMAP::getDist(int b, int e, ROUTE& r) {
    int x, y;
    double d = 0;
    if (1 == r && r[0].F() == r[0].T()) {     //乘坐线路=转乘线路：不用转乘
        d = GIS::ls[r[0].F()].dist(b, e);
        return d;
    }
    d = GIS::ls[r[0].F()].dist(b, r[0].S());
```

```
    y = (int)r - 1;              //转乘次数
    for (x = 0; x < y; x++)
        d += GIS::ls[r[x].T()].dist(r[x].S(), r[x + 1].S());
    d += GIS::ls[r[y].T()].dist(r[y].S(), e);
    return d;
}
//根据起点站 s 和终点站 t 找到最短距离 dist 若干线路 r，返回实际线路数
int TMAP::miniDist(int s, int t, double& dist, ROUTE(&r)[100])const {
    int k, u, v, w, x, y, z;
    int b = 0, e = 0;            //s:包含站点 s(站点数组下标）的起始线路（存放在 bls)数
    double dot[100]{};           //对应 r 的转乘距离
    int bls[20], els[20];     //e:包含站点 t(站点数组下标）的起始线路（存放在 els)数
    NODE rou;
    for (z = 0; z < GIS::nl; z++) {                    //寻找相关公交线路下标
        if (GIS::ls[z].has(s) != -1) if (b < 20) bls[b++] = z;
        if (GIS::ls[z].has(t) != -1) if (e < 20) els[e++] = z;
    }
    for (x = z = 0; x < b; x++)                         //b 个起始站点
        for (y = 0; y < e; y++) {                       //e 个到达站点
            rou = GIS::tra[bls[x]][els[y]];        //得到两线路的所有转乘线路
            w = GIS::tra(bls[x], els[y]);           //得到两线路的所有转乘线路数
            if (w == 0) continue;                       //本次起始站点的转乘线路数 w==0
            for (v = 0; v < w; v++) {
                u = TMAP::getDist(s, t, rou[v]);//得到转乘距离
                if (z == 0 || u < dot[0]) {         //若 dot 为空，或转乘次数比 dot[0]小
                    dot[0] = u;
                    r[0] = rou[v];
                    z = 1;
                }
                if (u == dot[0]) {                      //和已有线路转乘距离相同时，则插入
                    if (z == 100) return z;
                    dot[z] = u;
                    r[z] = rou[v];
                    z++;
                }
            }
```

```
            }
        dist = dot[0];    //升序插入:dot[0]到 dot[z-1]存储的距离相同
        return z;            //z 个乘坐方案
}

void TMAP::print()const {
        for (int x = 0; x < r; x++)
            for (int y = 0; y < c; y++) {
                printf("Node(%d,%d) has %d routes:\n", x, y, (int)(p[x * c + y]));
                p[x * c + y].print();
            }

}
//====================================organizations==============
organization::organization(const char* cname, int x, int y):name(cname)
{
        //this->name = name;
        this->x = x;
        this->y = y;
}
int& organization::X()
{
        return x;
}
int& organization::Y()
{
        return y;
}
const char* organization::NAME()
{
        return name;
}
//====================================GIS=======================================
===============
GIS::GIS() { obs++; }
GIS::GIS(const char* flstop, const char* flline,const char*florg) {
```

```
int    m, n, p, q, r, * s, * t;
FILE* fs, * fl,*forg;
fs = fopen(flstop, "r");
fl = fopen(flline, "r");
forg = fopen(florg, "r");
if (fs == 0 || fl == 0||forg==0) throw "File open error!";
org = new organization[100];
int cnt = 0;
while (1)
{
	char* s;
	int nowx, nowy;
	int t=fscanf(forg, "%s%d,%d", s, &nowx, &nowy);
	if (t == EOF)
		break;
	const char* now=s;
	org[cnt].NAME(now);
	org[cnt].X() = nowx;
	org[cnt++].Y() = nowy;
}
no = cnt;
fclose(forg);
numoforg = cnt;
fscanf(fs, "%d", &ns);
st = new STOP[ns];
for (m = 0; m < ns; m++) {
	fscanf(fs, "%d%d", &st[m].X(), &st[m].Y());
	st[m].N() = m + 1;		//公交线路编号从 1 开始
}
fclose(fs);
fscanf(fl, "%d", &nl);
s = new int[nl];			//每条线路的站点数
t = new int[100];			//每条线路的站点数不超过 100 站
for (m = 0; m < nl; m++) {
	fscanf(fl, "%d", &s[m]);
}
```

```
*(LINE**)&ls = new LINE[nl];
for (m = 0; m < nl; m++) {
    for (n = 0; n < s[m]; n++) {
        fscanf(fl, "%d", &t[n]);
        t[n]--;
    }
    ls[m] = LINE(m + 1, s[m], t);
}
fclose(fl);
for (m = 0; m < nl; m++) {    //构造 raw
    for (p = n = 0; n < nl; n++)
        if (m != n) p += GIS::ls[m].cross(GIS::ls[n]);
    if (p == 0) {
        printf("line %d does nott cross any line\n", m + 1);
        throw    "there is independent line";
    }
}
TMAP ra(nl, nl);
ROUTE a;
TRAN* u = new TRAN[100];
for (m = 0; m < nl; m++)
    for (n = 0; n < nl; n++)
    {
        if (m == n) {                        //本线路自身可在任一点转移
            u[0] = TRAN(m, n, -1);    //本线路自身可在任一点转移
            a = ROUTE(&u[0], 1);      //线路只有一次转乘
            ra(m, n, a);
            continue;
        }
        p = 0;                                //公交线路交点个数
        for (q = GIS::ls[m].NOFS() - 1; q >= 0; q--) {
            r = GIS::ls[m][q];
            if (GIS::ls[n].has(r) != -1)
            {//每个交点是一个转乘方式：新路线
                u[p] = TRAN(m, n, r);
                a = ROUTE(&u[p++], 1); //线路只有一次转乘
```

```
                        ra(m, n, a);
                    }
                }
            }
        tra = raw = ra;
        for (n = 2; n < nl; n++) {//构造闭包
            raw *= ra;
            tra += raw;
        }
        raw = ra;
        delete s;
        delete t;
        delete[]u;
        obs++;
}
GIS::~GIS() {
        obs--;

        if (obs) return;
        if (st) { delete[]st; *(STOP**)&st = 0; }
        if (ls) { delete[]ls; *(LINE**)&ls = 0; }
}
//根据步行起点和终点找到最少转乘次数 n 的若干线路 r，返回线路数
int GIS::miniTran(int fx, int fy, int tx, int ty, int& f, int& t, int& n, ROUTE(&r)[100]) {
        int m;
        double df, tf, dt, tt;
        f = 0;          //设离起点最近的站点 f
        df = sqrt((st[0].X() - fx) * (st[0].X() - fx) + (st[0].Y() - fy) * (st[0].Y() - fy));
        t = 0;          //设离终点最近的站点 t
        dt = sqrt((st[0].X() - tx) * (st[0].X() - tx) + (st[0].Y() - ty) * (st[0].Y() - ty));
        for (m = 1; m < GIS::ns; m++) {             //搜索离起点或终点最近的站点
            tf = sqrt((st[m].X() - fx) * (st[m].X() - fx) + (st[m].Y() - fy) * (st[m].Y() - fy));
            if (df > tf) { df = tf;   f = m; }     //离步行起点最近的站点，存在 f 中
            tt = sqrt((st[m].X() - tx) * (st[m].X() - tx) + (st[m].Y() - ty) * (st[m].Y() - ty));
            if (dt > tt) { dt = tt;   t = m; }
        }
```

```
    if (f == t) return 0;     //起点和终点相同，不用乘车
    return GIS::tra.miniTran(f, t, n, r);
}
//根据步行起点和终点找到最短距离 d 的若干线路 r,返回线路数
int GIS::miniDist(int fx, int fy, int tx, int ty, int& f, int& t, double& d, ROUTE(&r)[100]) {
    int m;
    double df, tf, dt, tt;
    f = 0;
    df = sqrt((st[0].X() - fx) * (st[0].X() - fx) + (st[0].Y() - fy) * (st[0].Y() - fy));
    t = 0;
    dt = sqrt((st[0].X() - tx) * (st[0].X() - tx) + (st[0].Y() - ty) * (st[0].Y() - ty));
    for (m = 1; m < GIS::ns; m++) {
        tf = sqrt((st[m].X() - fx) * (st[m].X() - fx) + (st[m].Y() - fy) * (st[m].Y() - fy));
        if (df > tf) { df = tf;    f = m; }
        tt = sqrt((st[m].X() - tx) * (st[m].X() - tx) + (st[m].Y() - ty) * (st[m].Y() - ty));
        if (dt > tt) { dt = tt;    t = m; }
    }
    if (f == t) return 0;     //起点和终点相同，不用乘车
    return GIS::tra.miniDist(f, t, d, r);
}
QtTipsDlgView.h:
#pragma once
#include<QTimer>
#include <QDialog>
#include "ui_QtTipsDlgView.h"

class QtTipsDlgView : public QDialog
{
    Q_OBJECT

public:
    QtTipsDlgView(const QString& msg, QWidget* parent = Q_NULLPTR);
    ~QtTipsDlgView();
    void startTimer(int);
private:
    Ui::QtTipsDlgView ui;
```

```
    QTimer* m_pTimer;
    void initFrame(const QString& msg);
};
```

QtTipsDlgView.cpp:

```
#include "QtTipsDlgView.h"

QtTipsDlgView::QtTipsDlgView(const QString& msg, QWidget* parent)
    : QDialog(parent)
{
    ui.setupUi(this);
    setWindowFlags(Qt::FramelessWindowHint | Qt::Tool | Qt::WindowStaysOnTopHint);
    setAttribute(Qt::WA_TranslucentBackground);

    initFrame(msg);

    m_pTimer = new QTimer(this);
    m_pTimer->setSingleShot(true);   //定时器只执行一次
    connect(m_pTimer, &QTimer::timeout, this, [=]() {this->close(); });
}

QtTipsDlgView::~QtTipsDlgView()
{
    if (this->m_pTimer != Q_NULLPTR)
    {
        this->m_pTimer->deleteLater();
    }
}

void QtTipsDlgView::startTimer(int ms)
{
    this->m_pTimer->start(ms);
}
void QtTipsDlgView::initFrame(const QString& msg)
{
    ui.labelMsg->setText(msg);
}
```

QtWidgetsFL.h:

```
#pragma once

#include <QWidget>
#include "ui_QtWidgetsFL.h"

class QtWidgetsFL : public QWidget
{
    Q_OBJECT
public:
    QtWidgetsFL(QWidget* parent = Q_NULLPTR);
    ~QtWidgetsFL();
    QGraphicsView* parnt;
    void myShow(QGraphicsView* p);
private:
    Ui::QtWidgetsFL ui;
private slots:
    void inputStop();
    void inputLine();
    void checkFile();
    void inputOrg();
};
```

QtWidgetsFL.cpp:

```
#include <QFileDialog>
#include <QMessageBox>
#include "QtWidgetsFL.h"
#include "logiclayer.h"
#include "twnlt.h"

//自定义的站点及线路输入界面
QtWidgetsFL::QtWidgetsFL(QWidget* parent)
    : QWidget(parent)
{
    ui.setupUi(this);
    connect(ui.pushButtonStop, SIGNAL(clicked()), this, SLOT(inputStop()), Qt::UniqueConnection);
    connect(ui.pushButtonLine, SIGNAL(clicked()), this, SLOT(inputLine()), Qt::UniqueConnection);
```

```
    connect(ui.pushButtonorg, SIGNAL(clicked()), this, SLOT(inputOrg()), Qt::UniqueConnection);
    connect(ui.pushButtonDone, SIGNAL(clicked()), this, SLOT(checkFile()), Qt::UniqueConnection);
    connect(ui.pushButtonDone, SIGNAL(clicked()),this,SLOT(close()),Qt::UniqueConnection);
}
void QtWidgetsFL::myShow(QGraphicsView* p) {
    parnt = p;
    show();
}
QtWidgetsFL::~QtWidgetsFL()
{
}
void QtWidgetsFL::inputStop() {
    ui.labelHits->setText("");
    QString fileName = QFileDialog::getOpenFileName(this, tr("Open File"), ".", tr("*.txt"));
    ui.textEditStop->setText(fileName);
}
void QtWidgetsFL::inputLine() {
    ui.labelHits->setText("");
    QString fileName = QFileDialog::getOpenFileName(this, tr("Open File"), "lines", tr("*.txt"));
    ui.textEditLine->setText(fileName);
}
void QtWidgetsFL::inputOrg()
{
    ui.labelHits->setText("");
    QString fileName = QFileDialog::getOpenFileName(this, tr("Open File"), ".", tr("*.txt"));
    ui.textEditorg->setText(fileName);
}

void QtWidgetsFL::checkFile() {
    QString   fs = ui.textEditStop->toPlainText();
    QString   fl = ui.textEditLine->toPlainText();
    QString   forg = ui.textEditorg->toPlainText();
    if (fs.isEmpty() && fl.isEmpty()&&forg.isEmpty()) {
        ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");
        ui.labelHits->setText(QString::fromLocal8Bit("操作提示：没有输入站点及线路文件路径！")); //不用
fromLocal8Bit 显示乱码
```

```
        ui.textEditStop->setFocus();
        return;
    }
    if (fs.isEmpty()) {
        ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");
        ui.labelHits->setText(QString::fromLocal8Bit("操作提示：没有输入站点文件路径！"));
        ui.textEditStop->setFocus();
        return;
    }
    if (fl.isEmpty()) {
        ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");
        ui.labelHits->setText(QString::fromLocal8Bit("操作提示：没有输入线路文件路径！"));
        ui.textEditLine->setFocus();
        return;
    }
    if (forg.isEmpty())
    {
        ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");
        ui.labelHits->setText(QString::fromLocal8Bit("操作提示：没有输入地名文件路径！"));
        ui.textEditLine->setFocus();
        return;
    }
    //处理站点文件
    ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");    //设置操作提示信息显示颜色
    ui.labelHits->setText(QString::fromLocal8Bit("操作提示：正在处理站点和线路文件..."));
    try {//读入站点及线路文件并初始化
        if (gis != nullptr) delete gis;
        std::string s1 = forg.toStdString();
        const char* ss = s1.c_str();
        gis = new GIS(fs.toStdString().c_str(), fl.toStdString().c_str(),ss);
        ((MyScene*)(parnt->scene()))->stopLines(parnt); //在背景地图上画出站点及公交线路
    }
    catch (...) {//读入或初始化失败
        gis = nullptr;
        close();
        QMessageBox::information(NULL, QString::fromLocal8Bit("操作提示"), QString::fromLocal8Bit("公
```

```
交站点或公交线路文件读入及初始化失败！"));
    }
    ui.labelHits->setText("");
    close();
}
```

MainWindows.h:

```
#pragma once

#include <QMainWindow>
#include<qcompleter.h>
#include "ui_MainWindow.h"
#include<Qlistview>
#include "logiclayer.h"
#include "twnlt.h"
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QStringList qstr,QWidget *parent = Q_NULLPTR);
    void FuzzySearch();
    void checkinput();
    ~MainWindow();

private:
    Ui::MainWindow ui;
    QStringList qstr;
    QString start;
    QString end;
};
```

MainWindows.cpp:

```
#include "MainWindow.h"

MainWindow::MainWindow(QStringList qstr,QWidget *parent)
    : QMainWindow(parent)
{
```

```
    this->qstr = qstr;
    ui.setupUi(this);
    FuzzySearch();
}

MainWindow::~MainWindow()
{
}
void MainWindow::FuzzySearch()
{

    ui.comboBox->addItems(qstr);
    ui.comboBox_2->addItems(qstr);
    ui.comboBox->setView(new QListView());
    ui.comboBox->setEditable(true);
    ui.comboBox->setLineEdit(ui.lineEdit);
    ui.comboBox->setMaxVisibleItems(5);//下拉列表显示 item 数
    ui.comboBox_2->setView(new QListView());
    ui.comboBox_2->setEditable(true);
    ui.comboBox_2->setMaxVisibleItems(5);
    ui.comboBox_2->setLineEdit(ui.lineEdit_2);
//      QString arrowImagePath = ":/res/combox.png";
//      ui->comboBox->setStyleSheet("QComboBox {font-family: \"Arial\"; font-size: 13px; padding: 3px 0x 3px
5px;}"
//            "QComboBox::drop-down {subcontrol-origin: padding; subcontrol-position: top right; width: 30 px;
border: 0px;}"
//            "QComboBox::down-arrow {image: url("+ arrowImagePath +");}");

    ui.comboBox->view()->setStyleSheet("QListView {font-family: \"Arial\"; font-size: 13px; outline: 0px;}"
        "QListView::item {padding: 3px 0x 3px 5px; border-width: 0px;}"
        "QListView::item:selected {background-color: rgb(74, 144, 226);}");
    QCompleter* pCompleter = new QCompleter(qstr, this);
    ui.lineEdit->setCompleter(pCompleter);
    pCompleter->setCaseSensitivity(Qt::CaseInsensitive);
    ui.comboBox->setCompleter(pCompleter);
    ui.lineEdit->clear();
```

```
    ui.comboBox_2->view()->setStyleSheet("QListView {font-family: \"Arial\"; font-size: 13px; outline: 0px;}"
        "QListView::item {padding: 3px 0x 3px 5px; border-width: 0px;}"
        "QListView::item:selected {background-color: rgb(74, 144, 226);}");
    QCompleter* pCompleter2 = new QCompleter(qstr, this);
    ui.lineEdit_2->setCompleter(pCompleter2);
    pCompleter2->setCaseSensitivity(Qt::CaseInsensitive);
    ui.comboBox_2->setCompleter(pCompleter2);
    ui.lineEdit_2->clear();
}
void MainWindow::checkinput()
{
    QString start = ui.lineEdit->text();
    int flag1 = 0, flag2 = 0;
    QString end = ui.lineEdit_2->text();
    //if (start.isEmpty() || start.isEmpty())
    //{
    //    ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");
    //    ui.labelHits->setText(QString::fromLocal8Bit("操作提示：没有输入站点及线路文件路径！")); //不用 fromLocal8Bit 显示乱码
    //    ui.textEditStop->setFocus();
    //    return;
    //}
    int i = 0;
    int pos1=0, pos2=0;
    QList<QString>::Iterator it=qstr.begin();
    for (; it != qstr.end(); it++)
    {
        if (*it == start)
        {
            pos1 = i;
        }
        else if (*it == end)
        {
            pos2 = i;
        }
        i++;
```

```
    }
    if (pos1 && pos2)
    {
        gis->end = pos1;
        gis->start = pos2;
        gis->flag = true;
    }
    else
    {

    }
}
```

twnlt.h:

```
#pragma once
#include <QtWidgets/QMainWindow>
#include<QGraphicsRectItem>
#include "ui_twnlt.h"
class QtWidgetsFL;
class MainWindow;
class twnlt : public QMainWindow
{
    Q_OBJECT

public:
    twnlt(QWidget* parent = Q_NULLPTR);
    ~twnlt();
private:
    Ui::twnltClass ui;
    QtWidgetsFL* fl;
    MainWindow* win;
    QTimer* m_Timer;
    QGraphicsItemGroup* gItem;
    void deleteItems();
protected:
    void closeEvent(QCloseEvent* event);
private slots:
```

```
    void loadmap();
    void closewnd();
    void zszc();
    void zdjl();
    void findinput();
};
class MyScene :public QGraphicsScene
{
public:
    explicit MyScene(QObject* parent = 0);
    void stopLines(QGraphicsView*);
protected:
    QGraphicsView* qgv;
    void mouseMoveEvent(QGraphicsSceneMouseEvent* event);
    void mousePressEvent(QGraphicsSceneMouseEvent* event);
public slots:
};
class MyItem :public QGraphicsRectItem {
    int cx, cy;//当前图元的坐标
    int cf;//标记左键或者右键，左键为 1，右键为 2
    int cs;//记录 bs 里存储的站点数目
    int bs[6];//记录与当前图元距离最近的站点编号
public:
    MyItem(int x, int y, int f);
    MyItem& operator<<(int s);
    int operator()(int x, int y);
    int& x();
    int& y();
    int& f();
    int& c();
    int& operator[](int);
    int checkAllStops();
    void mousePressEvent(QGraphicsSceneMouseEvent* event);
};
```

twnlt.cpp:

```
#include "twnlt.h"
```

```
#include "logiclayer.h"
#include "QtWidgetsFL.h"
#include "MainWindow.h"
#include <QDialog>
#include <QDesktopWidget>
#include <QApplication>
#include <QMessageBox>
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsEllipseItem>
#include "QtTipsDlgView.h"
using namespace std;
GIS* gis = nullptr;         //地图信息
bool route = false;            //未显示查询路径时
bool depart = false;       //未选取步行起点
bool arrive = false;       //未选取步行终点
MyItem::MyItem(int x, int y, int f) : QGraphicsRectItem(x - 3, y - 3, 7, 7), cx(x), cy(y), cf(f), cs(0) {
    //以下根据鼠标左键点击步行起点和鼠标右键点击步行终点设置不同画笔颜色
    QBrush qbrush(f == 1 ? QColor(210, 105, 45) : QColor(255, 170, 00));     //根据 f 设置颜色
    QPen qpens(qbrush, 4, Qt::SolidLine, Qt::SquareCap, Qt::BevelJoin); //图元点的画笔
    QGraphicsRectItem* item = this;
    item->setPen(qpens);
    item->setBrush(qbrush);
    //item->setFlag(QGraphicsItem::ItemIsMovable); //不选中：因为没有移动图元点的操作
    checkAllStops(); //检测所有站点，找出最近的站点存于 bs 中
}
int& MyItem::x() { return cx; }
int& MyItem::y() { return cy; }
int& MyItem::f() { return cf; }
int& MyItem::c() { return cs; }
int& MyItem::operator[](int x) {
    if (x < 0 || x >= cs) throw "subscription overflow for stops around departure point!";
    return bs[x];     //返回已检测的编号为 x 的站点编号
}
int MyItem::operator()(int x, int y) {    //仅用于检测两点距离远近，不开方
    return (x - cx) * (x - cx) + (y - cy) * (y - cy);
}
```

```
MyItem& MyItem::operator<<(int s) {
    if (s < 0 || s >= GIS::ns) return *this;
    int d = (*this)(GIS::st[s].X(), GIS::st[s].Y()); //d 为当前图元点到站点 s 的距离
    int m;                  //m 为当前图元点到先前已检测站点的距离
    if (cs == 0 || d < (m = (*this)(GIS::st[bs[0]].X(), GIS::st[bs[0]].Y()))) { //若 bs 没有元素即 cs==0，或距离站点 s 更近
        bs[0] = s;
        cs = 1;
        return *this;
    }
    if (d == m) {     //和已检测站点比，到标号为 s 的站点的距离相同时
        if (cs == 6) return *this;
        bs[cs] = s; //只保存和最近距离相同的站点
        cs++;           //距离相同的站点个数增加
    }
    return *this;
}
int MyItem::checkAllStops() {//检测所有站点，找出最近的站点存于 bs 中
    for (int c = 0; c < GIS::ns; c++)
        operator<<(GIS::st[c].N());
    return cs;                      //返回距离最近且相同的站点个数
}
void MyItem::mousePressEvent(QGraphicsSceneMouseEvent* event) {
    setSelected(true);          //当前图元被选中
    QGraphicsRectItem::mousePressEvent(event);
}
//定义自己的场景 MyScene，以便能够捕获鼠标或键盘事件
MyScene::MyScene(QObject* parent) : QGraphicsScene(parent)
{
    clearFocus();
    qgv = Q_NULLPTR;              //没有加载地图文件时
}
void MyScene::mouseMoveEvent(QGraphicsSceneMouseEvent* event) {
    //注意在其.ui 界面文件中，mouseTracking 必须勾选，否则不会出现此事件
    if (qgv == Q_NULLPTR) { //如果没有加载地图文件
        QGraphicsScene::mouseMoveEvent(event);
```

```
        return;
    }
    QPointF qpointf = event->scenePos();        //获取鼠标移动时的坐标
    for (int n = 0; n < gis->ns; n++) {
        if (fabs(gis->st[n].X() - qpointf.x()) < 8 && fabs(gis->st[n].Y() - qpointf.y()) < 8) {
            //以下提示信息必须使用 fromLocal8Bit()，否则中文提示会显示乱码
            QtTipsDlgView  dlg(QString::fromLocal8Bit(" 第 ") + QString::number(n + 1, 10, 0) +
QString::fromLocal8Bit("个公交站点."));
            //dlg.setAttribute(Qt::WA_ShowModal,true); 若调用 show()则需设置无边框,若调用 dlg.exec()则
不用此行，
            dlg.startTimer(1000);                    //设置悬停显示时间为 2 秒，时间到自动关闭 dlg
            QPointF m1 = qgv->mapToGlobal(QPoint(qpointf.x(), qpointf.y()));
            dlg.move(QPoint(m1.x(), m1.y()));
            dlg.exec();                              //显示站点提示信息
        }
    }
    for (int n = 0; n < gis->no; n++) {
        if (fabs(gis->org[n].X() - qpointf.x()) < 8 && fabs(gis->org[n].Y() - qpointf.y()) < 8) {
            QtTipsDlgView dlg(QString::fromLocal8Bit(gis->org[n].NAME()));
            dlg.startTimer(1000);
            QPointF m2 = qgv->mapToGlobal(QPoint(qpointf.x(), qpointf.y()));
            dlg.move(QPoint(m2.x(), m2.y()));
            dlg.exec();
        }
    }
    QGraphicsScene::mouseMoveEvent(event);       //回调基类鼠标事件
}
void MyScene::mousePressEvent(QGraphicsSceneMouseEvent* event)
{
    //qDebug("*********MyScene::mousePressEvent***************");
    if (qgv == Q_NULLPTR || route == true) {         //未加载地图及显示查询结果时，不响应鼠标按下事件
        QGraphicsScene::mouseMoveEvent(event);
        return;
    }
    QPointF qpointf = event->scenePos();        //获取鼠标坐标
    QList<QGraphicsItem*> listItem = items();
```

```
    int lb = 0;
    if (event->button() == Qt::LeftButton) { lb = 1; depart = true; }     //检查左键是否按下
    if (event->button() == Qt::RightButton) { lb = 2; arrive = true; }    //检查右键是否按下
    for (int i = listItem.length() - 1; i >= 0; i--) {
        MyItem* item = (MyItem*)listItem[i];
        if (item->f() == lb) {
            listItem.removeAt(i);
            delete item;
            break;
        }
    }
    addItem(new MyItem(qpointf.x(), qpointf.y(), lb));
    QGraphicsScene::mousePressEvent(event);                //回调基类鼠标事件
}
void MyScene::stopLines(QGraphicsView* parnt) {        //加载地图站点和线路
        //按视图 graphicsview 大小设置 scene 显示区域大大小
    QSize viewsize = parnt->size();                        //取得 graphicsview 视图区域大小
    MyScene* scene;
    if (parnt->scene() != Q_NULLPTR)     delete parnt->scene();
    scene = new MyScene(parnt);                            //创建 scene
    scene->setSceneRect(0, 0, viewsize.width(), viewsize.height());
    scene->qgv = parnt;
    //显示所有公交站点
    QBrush qbrush(QColor(255, 0, 0));
    QBrush qbrush1(QColor(0,255,0));//设置颜色
    QPen qpens(qbrush, 4, Qt::SolidLine, Qt::SquareCap, Qt::BevelJoin); //站点的笔
    for (int n = 0; n < gis->ns; n++) {
        scene->addEllipse(gis->st[n].X(), gis->st[n].Y(), 6, 6, qpens, qbrush);
    }
    //显示所有线路
    QPen qpenl(qbrush, 3, Qt::SolidLine, Qt::SquareCap, Qt::BevelJoin); //线路的笔
    for (int n = 0; n < gis->nl; n++) {
        LINE& line = gis->ls[n];
        int stops = line.NOFS();
        for (int m = 1; m < stops; m++) {
            STOP s = gis->st[line[m - 1]];
```

```
                STOP t = gis->st[line[m]];
                QLineF ql = QLineF(s.X(), s.Y(), t.X(), t.Y());
                scene->addLine(ql, qpenl);
            }
        }
        QPen qpenorg(qbrush1,5,Qt::SolidLine,Qt::SquareCap,Qt::BevelJoin);
        for (int n = 0; n < gis->numoforg; n++)
        {
            scene->addEllipse(gis->org[n].X(), gis->org[n].Y(), 6, 6, qpenorg, qbrush1);
        }
        parnt->setScene(scene);
        parnt->show();
}
////////////////////////////////////////////////////////////////////////////////
twnlt::twnlt(QWidget* parent)
        : QMainWindow(parent)
{
        ui.setupUi(this);
        fl = Q_NULLPTR;
        gItem = Q_NULLPTR;
        win = Q_NULLPTR;
        m_Timer = new QTimer(this);
        m_Timer->setSingleShot(true);                    //定时器只执行一次
        connect(ui.action_open, SIGNAL(triggered(bool)), this, SLOT(loadmap()));
        connect(ui.action_exit, SIGNAL(triggered(bool)), this, SLOT(closewnd()));
        connect(ui.action_zszc, SIGNAL(triggered(bool)), this, SLOT(zszc()));
        connect(ui.action_zdjl, SIGNAL(triggered(bool)), this, SLOT(zdjl()));
        connect(ui.action_find, SIGNAL(triggered(bool)), this, SLOT(findinput()));
        //以下 Lambda 表达式可以用自动类型推导代替,或用非成员函数的地址代替
        connect(m_Timer, &QTimer::timeout, this, [=]() {
            QList<QGraphicsItem*> listItem = ui.graphicsView->scene()->items();
            deleteItems();          //查询结果显示时间一到，就删除场景的所有图元
            route = false;          //查询结果显示完毕，可以重新选取步行起点或终点
            });
}
twnlt::~twnlt() {
```

```
    if (fl != Q_NULLPTR) {
        fl->hide();
        delete fl;
        fl = Q_NULLPTR;
        delete m_Timer;
        deleteItems();
        delete    gis;
    }
}
void twnlt::closewnd() {
    if (ui.action_exit->isChecked() == false) return; //鼠标点击一次触发两次，第二次触发直接返回
    ui.action_exit->setChecked(false);   //鼠标第一次触发设置状态为 false,防止第 2 次触发进入
    if (fl != Q_NULLPTR) {
        fl->hide();
        delete fl;
        fl = Q_NULLPTR;
    }
    close();
}
void twnlt::closeEvent(QCloseEvent* event)
{
    if (fl != Q_NULLPTR) {
        fl->hide();
        delete fl;
        fl = Q_NULLPTR;
    }
}
void twnlt::loadmap() {

    if (ui.action_open->isChecked() == false) return; //鼠标点击触发两次，第二次触发直接返回
    ui.action_open->setChecked(false);   //鼠标第一次触发设置状态为 false,防止第 2 次触发进入
    if (fl != Q_NULLPTR) {    //如果先前打开过站点及线路输入窗口
        fl->show();               //则直接显示该窗口
        return;
    }
    arrive = depart = false;    //此时未选取步行起点或终点
```

```
    fl = new QtWidgetsFL(); //如果以前没有打开过站点及线路输入窗口
    fl->setWindowFlags(Qt::WindowStaysOnTopHint); //设置最外层显示
    fl->myShow(ui.graphicsView);
}
void twnlt::findinput()
{
    if (ui.action_find->isChecked() == false) return;
    ui.action_find->setChecked(false);
    if (win != Q_NULLPTR)
    {
        win->show();
        return;
    }
    arrive = depart = false;
    QStringList q;
    for (int n = 0; n < gis->numoforg; n++)
    {
        const char* ch= gis->org[n].NAME();
        QString str = QString::fromLocal8Bit(ch);
        q << str;
    }
    win = new MainWindow(q);
    win->show();
}
void twnlt::deleteItems() { //删除场景的所有图元
    if (gItem == Q_NULLPTR) return;
    ui.graphicsView->scene()->removeItem(gItem);
    for (int i = gItem->childItems().size() - 1; i >= 0; i--) {
        QGraphicsItem* item = (gItem->childItems())[i];
        gItem->removeFromGroup(item);
        delete item;
    }
    delete gItem;
    gItem = Q_NULLPTR;
}
void twnlt::zszc() {
```

```
//先计算最少转乘的路径,先获得起点坐标和终点坐标
if (ui.action_zszc->isChecked() == false) return; //鼠标点击一次触发两次，第二次触发直接返回
ui.action_zszc->setChecked(false);  //鼠标第一次触发设置状态为 false,防止第 2 次触发进入
QList<QGraphicsItem*> listItem;
if ((depart && arrive) == false) return;//若没有选中步行起点和终点，则返回
listItem = ui.graphicsView->scene()->items();
MyItem* itemDepart = (MyItem*)listItem[0];
MyItem* itemArrive = (MyItem*)listItem[1];
if (itemDepart->f() != 1) {              //若不是步行起点，则交换
    itemDepart = (MyItem*)listItem[1];
    itemArrive = (MyItem*)listItem[0];
}
//开始组建图元组：用于显示转乘方案的路径
QGraphicsEllipseItem* myEItem;
QGraphicsLineItem* myLItem;
MyScene* scene = (MyScene*)(ui.graphicsView->scene());
QBrush qbrush(QColor(0, 0, 255));//设置颜色
QPen qpens(qbrush, 4, Qt::SolidLine, Qt::SquareCap, Qt::BevelJoin); //站点的笔
QPen qpenl(qbrush, 3, Qt::SolidLine, Qt::SquareCap, Qt::BevelJoin); //线路的笔
route = true;                    //进入查询路径显示期间，不响应步行起点与终点选取
int c, n = 0;                    //c 为可行转乘方案数，n 为最少转乘次数
ROUTE r[100];                    //一次查询，最多返回 100 条可行转乘方案
for (int d = 0; d < itemDepart->c(); d++) {  //接近起点的所有公交站点
    int s = (*itemDepart)[d]; //获得起点站点编号 s
    for (int a = 0; a < itemArrive->c(); a++) { //接近终点的所有公交站点
        int t = (*itemArrive)[a];            //获得终点站点编号 t
        if (s == t) {                        //起点站和终点站相同不用转乘
            deleteItems();                   //删除先前的转乘方案路径显示
            gItem = new QGraphicsItemGroup();
            myEItem = new QGraphicsEllipseItem(itemDepart->x(), itemDepart->y(), 6, 6);
            myEItem->setPen(qpens);
            myEItem->setBrush(qbrush);
            gItem->addToGroup(myEItem);      //步行起点的位置
            myLItem = new QGraphicsLineItem(itemDepart->x(), itemDepart->y(), itemArrive->x(),
itemArrive->y());
            myLItem->setPen(qpenl);
```

```
                gItem->addToGroup(myLItem);              //到步行终点的路径
                myEItem = new QGraphicsEllipseItem(itemArrive->x(), itemArrive->y(), 6, 6);
                myEItem->setPen(qpens);
                myEItem->setBrush(QBrush(QColor(255, 170, 00)));
                gItem->addToGroup(myEItem);              //步行终点的位置
                scene->addItem(gItem);
                ui.graphicsView->setScene(scene);
                continue;
            }
            c = GIS::tra.miniTran(s, t, n, r);           //得到的转乘方案数
            for (int m = 0; m < c; m++) {                //对于第 m 个转乘方案即 route=r[m], 转乘次数都为
n
                deleteItems();
                gItem = new QGraphicsItemGroup();
                myEItem = new QGraphicsEllipseItem(itemDepart->x(), itemDepart->y(), 6, 6);
                myEItem->setPen(qpens);
                myEItem->setBrush(qbrush);
                gItem->addToGroup(myEItem);              //步行起点的位置
                int fr = s, to = t;                      //起始站 fr 与终点站 to
                int fm, tm;                              //已乘线路 fm 与 z 转乘线路 tm
                int bg, ed;                              //线路中的起始站点序号 bg,终止站点序号 ed
                myLItem = new QGraphicsLineItem(itemDepart->x(), itemDepart->y(), gis->st[s].X(),
gis->st[s].Y());
                myLItem->setPen(qpenl);
                gItem->addToGroup(myLItem);              //到起点站
                if (n == 1 && r[m][0].S() == -1) {   //即从 i 路到 i 路(此时 S()==-1)不用转乘
                    fm = r[m][0].F();                    //已乘线路序号 fm
                    bg = GIS::ls[fm].has(fr);            //起始站点在线路中的序号
                    ed = GIS::ls[fm].has(to);            //终止站点在线路中的序号
                    if (bg > ed) { tm = bg; bg = ed; ed = tm; }
                    for (int y = bg; y < ed; y++)    //从起始站点下一序号到终止站点序号
                    {
                        fr = GIS::ls[fm][y];             //得到该站点序号对应的站点编号
                        to = GIS::ls[fm][y + 1];         //得到该站点序号对应的站点编号
                        myLItem = new QGraphicsLineItem(GIS::st[fr].X(), GIS::st[fr].Y(), GIS::st[to].X(),
GIS::st[to].Y());
```

```
                myLItem->setPen(qpenl);
                gItem->addToGroup(myLItem); //到下一站的路径
            }
        }
        else {
            for (int y = 0; y < n; y++)        //对于每个转乘
            {
                fm = r[m][y].F();                  //对于每个转乘的起始线路
                bg = GIS::ls[fm].has(fr);
                to = r[m][y].S();                  //对于起始线路的转乘站点
                ed = GIS::ls[fm].has(to);
                if (bg > ed) { tm = bg; bg = ed; ed = tm; }
                for (int u = bg; u < ed; u++)   //从起始站点下一序号到终止站点序号
                {
                    int ff = GIS::ls[fm][u];       //得到该站点序号对应的站点编号
                    int tt = GIS::ls[fm][u + 1];   //得到该站点序号对应的站点编号
                    myLItem = new QGraphicsLineItem(GIS::st[ff].X(), GIS::st[ff].Y(),
GIS::st[tt].X(), GIS::st[tt].Y());
                    myLItem->setPen(qpenl);
                    gItem->addToGroup(myLItem); //到下一站的路径
                }
                fr = to;                                //作为下一起点
            }
            fm = r[m][n - 1].T();                       //对于最后乘坐的线路
            bg = GIS::ls[fm].has(fr);
            ed = GIS::ls[fm].has(t);
            if (bg > ed) { tm = bg; bg = ed; ed = tm; }
            for (int u = bg; u < ed; u++)           //从起始站点下一序号到终止站点序号
            {
                int ff = GIS::ls[fm][u];            //得到该站点序号对应的站点编号
                int tt = GIS::ls[fm][u + 1];       //得到该站点序号对应的站点编号
                myLItem = new QGraphicsLineItem(GIS::st[ff].X(), GIS::st[ff].Y(), GIS::st[tt].X(),
GIS::st[tt].Y());
                myLItem->setPen(qpenl);
                gItem->addToGroup(myLItem);        //到下一站的路径
            }
```

```
                }
                myLItem = new QGraphicsLineItem(GIS::st[t].X(), GIS::st[t].Y(), itemArrive->x(), itemArrive->y());
                myLItem->setPen(qpenl);
                gItem->addToGroup(myLItem);                    //到步行终点的路径
                myEItem = new QGraphicsEllipseItem(itemArrive->x(), itemArrive->y(), 6, 6);
                myEItem->setPen(qpens);
                myEItem->setBrush(QBrush(QColor(255, 170, 00)));
                gItem->addToGroup(myEItem);                    //步行终点的位置
                scene->addItem(gItem);
                ui.graphicsView->setScene(scene);
            }
        }
    }
    this->m_Timer->start(5000);   //展示查询的路径结果 5 秒
}
void twnlt::zdjl() {
    //先计算最短距离的路径,先获得起点坐标和终点坐标
    if (ui.action_zdjl->isChecked() == false) return; //鼠标点击一次触发两次，第二次出发直接返回
    ui.action_zdjl->setChecked(false);  //鼠标第一次出发，设置状态为 false,防止第 2 次出发进入
    QList<QGraphicsItem*> listItem;
    if ((depart && arrive) == false) return;
    listItem = ui.graphicsView->scene()->items();
    MyItem* itemDepart = (MyItem*)listItem[0];
    MyItem* itemArrive = (MyItem*)listItem[1];
    if (itemDepart->f() != 1) { //若不是起点，则交换
        itemDepart = (MyItem*)listItem[1];
        itemArrive = (MyItem*)listItem[0];
    }
    //开始组建图元组：用于显示转乘方案的路径
    QGraphicsEllipseItem* myEItem;
    QGraphicsLineItem* myLItem;
    MyScene* scene = (MyScene*)(ui.graphicsView->scene());
    QBrush qbrush(QColor(0, 0, 255));//设置颜色
    QPen qpens(qbrush, 4, Qt::SolidLine, Qt::SquareCap, Qt::BevelJoin); //站点的笔
    QPen qpenl(qbrush, 3, Qt::SolidLine, Qt::SquareCap, Qt::BevelJoin); //线路的笔
```

```
    route = true;                                   //进入查询路径显示期间，不响应步行起点与终点选取
    int c = 0;                                      //c 为可行线路数
    double dist = 0;                                //dist 为最短距离
    ROUTE r[100];
    for (int d = 0; d < itemDepart->c(); d++) {    //接近起点的所有公交站点
        int s = (*itemDepart)[d];                  //获得起点站点编号 s
        for (int a = 0; a < itemArrive->c(); a++) {    //接近终点的所有公交站点
            int t = (*itemArrive)[a];      //获得终点站点编号 t
            if (s == t) {                          //起点站和终点站相同不用转乘
                deleteItems();                     //删除先前的转乘方案路径显示
                gItem = new QGraphicsItemGroup();
                myEItem = new QGraphicsEllipseItem(itemDepart->x(), itemDepart->y(), 6, 6);
                myEItem->setPen(qpens);
                myEItem->setBrush(qbrush);
                gItem->addToGroup(myEItem);//步行起点的位置显示
                myLItem = new QGraphicsLineItem(itemDepart->x(), itemDepart->y(), itemArrive->x(),
itemArrive->y());
                myLItem->setPen(qpenl);
                gItem->addToGroup(myLItem);//到步行终点的路径
                myEItem = new QGraphicsEllipseItem(itemArrive->x(), itemArrive->y(), 6, 6);
                myEItem->setPen(qpens);
                myEItem->setBrush(QBrush(QColor(255, 170, 00)));
                gItem->addToGroup(myEItem);//步行终点的位置
                scene->addItem(gItem);
                ui.graphicsView->setScene(scene);
                continue;
            }
            c = GIS::tra.miniDist(s, t, dist, r);   //得到的转乘方案数 c
            for (int m = 0; m < c; m++) {      //对于第 m 个转乘方案即 route=r[m],最短距离都为 dist
                deleteItems();
                gItem = new QGraphicsItemGroup();
                myEItem = new QGraphicsEllipseItem(itemDepart->x(), itemDepart->y(), 6, 6);
                myEItem->setPen(qpens);
                myEItem->setBrush(qbrush);
                gItem->addToGroup(myEItem);        //步行起点的位置显示
                int fr = s, to = t;                    //起始站 fr 与终点站 to
```

```
                int fm, tm;                                 //已乘线路 fm 与 z 转乘线路 tm
                int bg, ed;                                 //线路中的起始站点序号 bg,终止站点序号 ed
                myLItem = new QGraphicsLineItem(itemDepart->x(), itemDepart->y(), gis->st[s].X(),
gis->st[s].Y());
                myLItem->setPen(qpenl);
                gItem->addToGroup(myLItem);         //到起点站的路径显示
                int n = r[m];                               //当前 route 中的转乘次数
                if (n == 1 && r[m][0].S() == -1) { //即从 i 路到 i 路(此时 S()==-1)不用转乘
                    fm = r[m][0].F();                       //已乘线路序号 fm
                    bg = GIS::ls[fm].has(fr);       //起始站点在线路中的序号
                    ed = GIS::ls[fm].has(to);       //终止站点在线路中的序号
                    if (bg > ed) { tm = bg; bg = ed; ed = tm; }
                    for (int y = bg; y < ed; y++) //从起始站点下一序号到终止站点序号
                    {
                        fr = GIS::ls[fm][y];        //得到该站点序号对应的站点编号
                        to = GIS::ls[fm][y + 1];//得到该站点序号对应的站点编号
                        myLItem = new QGraphicsLineItem(GIS::st[fr].X(), GIS::st[fr].Y(), GIS::st[to].X(),
GIS::st[to].Y());
                        myLItem->setPen(qpenl);
                        gItem->addToGroup(myLItem);//到下一站
                    }
                }
                else {
                    for (int y = 0; y < n; y++) //对于每个转乘画出路径
                    {
                        fm = r[m][y].F();               //对于每个转乘的起始线路
                        bg = GIS::ls[fm].has(fr);
                        to = r[m][y].S();               //对于起始线路的转乘站点
                        ed = GIS::ls[fm].has(to);
                        if (bg > ed) { tm = bg; bg = ed; ed = tm; }
                        for (int u = bg; u < ed; u++)       //从起始站点下一序号到终止站点序号
                        {
                            int ff = GIS::ls[fm][u];        //得到该站点序号对应的站点编号
                            int tt = GIS::ls[fm][u + 1];//得到该站点序号对应的站点编号
                            myLItem = new QGraphicsLineItem(GIS::st[ff].X(), GIS::st[ff].Y(),
GIS::st[tt].X(), GIS::st[tt].Y());
```

```
                    myLItem->setPen(qpenl);
                    gItem->addToGroup(myLItem); //到下一站的路径
                }
                fr = to;                                    //作为下一起点
            }
            fm = r[m][n - 1].T();                           //对于最后乘坐的线路
            bg = GIS::ls[fm].has(fr);
            ed = GIS::ls[fm].has(t);
            if (bg > ed) { tm = bg; bg = ed; ed = tm; }
            for (int u = bg; u < ed; u++)                   //从起始站点下一序号到终止站点序号
            {
                int ff = GIS::ls[fm][u];                    //得到该站点序号对应的站点编号
                int tt = GIS::ls[fm][u + 1];                //得到该站点序号对应的站点编号
                myLItem = new QGraphicsLineItem(GIS::st[ff].X(), GIS::st[ff].Y(), GIS::st[tt].X(),
GIS::st[tt].Y());
                myLItem->setPen(qpenl);
                gItem->addToGroup(myLItem);                 //到下一站的路径
            }
        }
        myLItem = new QGraphicsLineItem(GIS::st[t].X(), GIS::st[t].Y(), itemArrive->x(),
itemArrive->y());
        myLItem->setPen(qpenl);
        gItem->addToGroup(myLItem);                         //到步行终点的路径
        myEItem = new QGraphicsEllipseItem(itemArrive->x(), itemArrive->y(), 6, 6);
        myEItem->setPen(qpens);
        myEItem->setBrush(QBrush(QColor(255, 170, 00)));
        gItem->addToGroup(myEItem);                         //步行终点的位置显示
        scene->addItem(gItem);
        ui.graphicsView->setScene(scene);
    }
  }
 }
 this->m_Timer->start(5000);   //展示查询的路径结果 5 秒
}
```